2001-09

Enhancing network communication in NPSNET-V virtual Environments using XML : Described dynamic Behavior (DBP) Protocols

Fischer, William D.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

ENHANCING NETWORK COMMUNICATION IN NPSNET-V VIRTUAL ENVIRONMENTS USING XML-DESCRIBED DYNAMIC BEHAVIOR (DBP) PROTOCOLS

by

William D. Fischer

September 2001

Thesis Advisor:	Don McGregor
Co-Advisor:	Don Brutzman

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2001	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Enhancing Network Communication in NPSNET-V Virtual Environments using XML-Described Dynamic Behavior (DBP) Protocols			**5. FUNDING NUMBERS**
6. AUTHOR William D. Fischer			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			**12b. DISTRIBUTION CODE** A

13. ABSTRACT

The existing component protocols, as well as new protocols introduced at runtime into NPSNET-V are written in their native programming language. As a result, they require authoring and compiling by a trained programmer. The long time frame required to change or introduce new protocols into NPSNET-V, a dynamically extensible virtual environment, detracts from the dynamicism of the virtual environment. Networking optimization thresholds to support NPSNET-V needed to be determined to ensure that the networking is performed efficiently, and system resources to other systems, such as graphics rendering, are maximized.

This thesis implements component protocols described using Extensible Markup Language (XML) into NPSNET-V. These protocols are created with different fidelity resolutions for each protocol, which can be swapped at runtime based on the network state. Network testing was performed to find the ideal maximum packet rates based on the impact on CPU utilization and packet loss. By using XML, non-programmers can edit protocols for inclusion in a simulation at runtime.

Important contributions include adding protocols to NPSNET-V with high-resolution and low-resolution versions, described by XML documents. Basic network optimization is added to NPSNET-V to take advantage of the protocols' resolution switching ability. The network testing revealed a linear correlation between the packet sending rate and CPU utilization, and a polynomial correlation between the packet sending rate and percentage packet loss.

14. SUBJECT TERMS Network Monitoring, Virtual Environments, NPSNET, VRTP, Dynamic Behavior Prtocol (DBP), XML			**15. NUMBER OF PAGES** 136
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

ENHANCING NETWORK COMMUNICATION IN NPSNET-V VIRTUAL ENVIRONMENTS USING XML DESCRIBED DYNAMIC BEHAVIOR (DBP) PROTOCOLS

William D. Fischer
Major, United States Army
B.S., College of William and Mary, 1989

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 2001

Author: _____
William D. Fischer

Approved by: _____
Don McGregor, Thesis Advisor

_____
Don Brutzman, Co-Advisor

_____
Chris Eagle, Chair
Computer Science Department

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The existing component protocols, as well as new protocols introduced at runtime into NPSNET-V are written in their native programming language. As a result, they require authoring and compiling by a trained programmer. The long time frame required to change or introduce new protocols into NPSNET-V, a dynamically extensible virtual environment, detracts from the dynamicism of the virtual environment. Networking optimization thresholds to support NPSNET-V needed to be determined to ensure that the networking is performed efficiently, and system resources to other systems, such as graphics rendering, are maximized.

This thesis implements component protocols described using Extensible Markup Language (XML) into NPSNET-V. These protocols are created with different fidelity resolutions for each protocol, which can be swapped at runtime based on the network state. Network testing was performed to find the ideal maximum packet rates based on the impact on CPU utilization and packet loss. By using XML, non-programmers can edit protocols for inclusion in a simulation at runtime.

Important contributions include adding protocols to NPSNET-V with high-resolution and low-resolution versions, described by XML documents. Basic network optimization is added to NPSNET-V to take advantage of the protocols' resolution switching ability. The network testing revealed a linear correlation between the packet sending rate and CPU utilization, and a polynomial correlation between the packet sending rate and percentage packet loss.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

To Don McGregor, I thank you for your guidance, wisdom, and enthusiasm throughout this project. Your efforts have given me an invaluable learning experience. To Don Brutzman, thank you for sharing your insight and perspective during this project. You have helped me to maintain the proper focus throughout this endeavor.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND

This thesis investigates how the Extensible Markup Language (XML) can be used to describe application-specific networking protocols for use in the NPSNET-V Virtual Environment. Specifically, it describes design of the NPS Dynamic-Behavior-Protocol (DBP) protocols, which are multicast / unicast capable and can be added at runtime to the distributed operating environment. With XML-described packet payloads for these protocols, supporting configuration documents can concisely and clearly describe different variations of these protocols and can also enable switching between them at runtime.

Protocols described by XML-based DBP documents can improve network performance for large-scale networked virtual environments, since they can be tailored to best support the requirements for a particular environment. These improvements can be made rapidly and adaptively at runtime. Furthermore, protocols described in XML instead of an imperative programming language (such as Java or C++) can be modified by an adept nonprogrammer or even software agents, instead of requiring tedious debugging and recompilation by extensively trained computer programmers.

B. MOTIVATION

NPSNET-V is a research project to investigate dynamically extensible, large-scale virtual environments, including protocols that can be introduced into the environment at run-time. A limitation of current systems is that present-day networking protocols can't be altered without rewriting and recompiling source code, then stopping and restarting the system. Such constraints on protocol development prevent proper composability,

testing, monitoring and thus generally detracts from the NPSNET-V goal of runtime extensibility.

In contrast, a DBP implementation that can run, test, compare and improve continuously is expected to enhance the extensibility of networked virtual worlds, as well as adaptively optimize network performance. A DBP implementation can further enable computer-driven runtime optimization, thus providing a structure to make network optimization experiments easier to conduct.

C. OBJECTIVES

The goal of this thesis is to create DBP protocols that support communication between entities whose prototypes are runtime-created, for use in the NPSNET-V Research Project. These network protocols will facilitate exchanging state information between objects in the network and can be written in XML to facilitate flexibility, ease-of-use to the end user, and rapid creation at runtime. These protocols will support network optimization by being able to switch between different source documents that describe network protocols with different resolutions, again adaptively at runtime.

In NPSNET-V, a variety of software components including network protocols can be dynamically loaded. This thesis examines adding DBP protocols that can be loaded at runtime in the same manner as the pre-existing networking protocols written in Java. These DBP protocols will initially have the same semantics, or functionality, as the existing protocols that are described in Java. By changing the syntax of the XML source documents, we can affect changes in both the syntax and the semantics of the protocol at runtime.

All networking protocols are uploaded to a server, and a pointer to a protocol's location is posted to a LDAP server. When a participating machine first encounters an unknown protocol, it downloads the corresponding Java byte code associated with that protocol from the server pointed to by the LDAP server.

The network message-format descriptions for DBP entity communications are initially written in XML and posted to a web server. This byte code contains a URL that points to an XML description of the packet format. The Java byte code parses the XML file to discover the names and positions of the fields in the packet. The Java protocol code then uses this information to parse fields and retrieve data from the packet.

If changes need to be made to the format of a packet, then fields can be added, removed or modified in the XML packet description and the new description posted to a web server. Based on the new description, different collections of data can be passed in the network packets. Since the packet layouts are written in XML, protocols can be quickly distributed to participating hosts and written, posted and incorporated during execution of a simulation. Writing XML documents, which describe packet contents does not require a programmer, an approach that supports and enhances the dynamicism and authorability of NPSNET-V. XML documents are similar to HTML in many respects and can be created or modified with a text editor. Thus an XML document describing a protocol can be quickly generated this way and incorporated into a simulation at runtime.

These XML-described protocols can be implemented in a manner that supports extensibility since the payload characteristics of protocols can be easily changed at runtime to adapt to specific requirements, such as varying network load. If a new

protocol definition is created, the XML document is parsed by an XML parsing utility, and the new protocol description is included in the running process. This capability to rapidly create and switch between protocols enables performing dynamic network optimization, in ways that are tuned by the diverse needs of the application.

To demonstrate the effectiveness of these XML-described protocols, example protocols are each constructed with one XML document supporting a high-resolution packet payload and another document describing a low-resolution packet format. In support of this demonstration a single network optimization is implemented that takes advantage of this dynamic protocol-switching capability. This networking optimization includes maintaining information about the ongoing state of the network, and also has the ability to adjust protocols on demand for global optimization. Furthermore, experimental tests are conducted to determine networking points of failure to determine when the protocol switching is best performed. The two metrics leading to possible failure conditions that are examined here are CPU utilization and packet loss. Summarizing, the thesis objectives are straightforward and summarized in Figure 1.

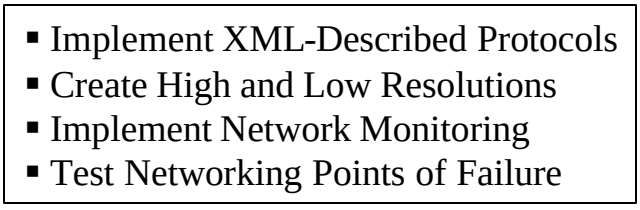

Figure 1. Thesis Objectives

D. THESIS ORGANIZATION

The chapters of this thesis are organized as follows. Chapter II provides the background for this thesis, including an overview of subjects considered in this thesis and related work necessary for understanding context of the project. Chapter III presents an NPSNET-V overview and a discussion of networking in NPSNET-V. Chapter IV introduces the strategy used to implement the DBP protocols, to implement a network monitor, and to perform the network testing. Chapter V provides the results and analysis of data collection and testing, then discusses tradeoffs that must be made to optimize the protocols. Chapter VI presents thesis conclusions and provides recommendations for future work.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND AND RELATED WORK

A. INTRODUCTION

Networked virtual environments comprise an active and contemporary research topic that is continuing to develop and improve. Like most other topics in Computer Science, efforts in this area build on the progress already made.

Design patterns have been published that provide solutions to common software-design challenges. Standard procedures have also been developed to construct network testing plans. New technologies have been developed, such as XML, which provide opportunities to apply new solutions to old challenges. Concurrent and recent work in the same area impacts design considerations, which must consider interoperability. This chapter presents a variety of background material that directly influences this thesis.

B. DESIGN PATTERNS

> Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice. (Alexander, 77)

1. Design Patterns Motivation

Design patterns provide a common strategy for solving complex problems. By naming these patterns, programmers gain a common vocabulary by which to discuss them. Patterns focus on specific object-oriented design challenges. Individual patterns indicate when they apply, what impact their use has on a project, and what the consequences of their use are. Patterns identify classes (and / or interfaces), class instances, roles, collaborations, and responsibilities (Gamma, 95).

Several design patterns are used in NPSNET-V. The most important patterns in this thesis include the singleton, observer, and model / view / controller patterns.

2. Singleton Pattern

The singleton pattern ensures that the class following the pattern has only one instance, and that there is a global means of accessing that instance (Gamma, 95). The singleton pattern is usually used for a class controlling a centrally managed resource (Grand, 98).

In a Java implementation, the singleton class has a static variable referring to the single instance of the class. The class provides access to this instance with a static method, which returns a reference to this instance. All of the class's constructors are private, to prevent instantiating another instance of the class (Grand, 98).

3. Observer Pattern

The observer pattern defines "… a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically" (Gamma, 95). This pattern uses subjects and observers, with a subject able to have many observers. When the subject experiences a change in state, it notifies its observers, who in turn query the subject for what the changes are (Gamma, 95).

In a Java implementation, the subject usually passes a self-reference as a parameter to the observer. This process generally uses an interface for the method call, to enable run-time registration of observers. The subject also provides access to its state attributes for the observer to act on the changes (Grand, 98).

4. Model / View / Controller (MVC) Pattern

The Model / View / Controller (MVC) pattern was originally a combination of classes designed to build user interfaces in Smalltalk-80. The model object is the application object that contains persistent state variables, the view object is the graphical representation of the model, and the controller object defines the model's reaction to user input. By loosely coupling these three components, users can have multiple views for an object, and also change a view to better reflect the state of the model. Likewise, there can be different means of user input. Following the MVC pattern enables each component to make changes based on information from another component, without have to know the inner workings of the other two components (Gamma, 95).

C. NETWORK MONITORING

1. Network Performance Evaluations

Network performance evaluations are used to determine how well a system is performing, and whether any improvements are necessary. The following points in Figure 2 are from the book "The Art of Computer Systems Performance Analysis" (Jain, 91) and outline the steps in performing such a study.

- State the goals of the study and define the system boundaries
- List system services and possible outcomes
- Select performance metrics
- List system and workload parameters
- Select factors and their values
- Select evaluation techniques
- Select the workload
- Design the experiments
- Analyze and interpret the data
- Present the results. Start over, if necessary

Figure 2. Network Performance Evaluation Steps (Jain, 91)

2. Network Evaluation Techniques

The three evaluation techniques used by this thesis to evaluate networks are analytical modeling, simulation and measurement. Analytical evaluation is best used when time is short and resources are low. Analytical evaluation is usually performed mathematically and does not include any data collection. Simulation evaluation involves building a model of the network being tested and collecting data. Measurement evaluation is performed when gathering actual data from the network being tested. Since testing is susceptible to errors, any network evaluation of one type needs to be verified by performing an evaluation of another type (Jain, 91).

This thesis uses measurement evaluation through network testing. The metrics collected focus on the impact that packet sending rates have on CPU utilization and packet loss. Performance thresholds are then determined during this testing. This test implementation involves reporting the network state to the Area of Interest Manager (AOIM). The thresholds are then validated during a simulation of a virtual environment running on top of NPSNET-V.

D. EXTENSIBLE MARKUP LANGUAGE (XML)

Extensible Markup Language (XML) is a meta-language used to define structured-data documents for other languages. XML looks similar to HTML, but is very different and much more powerful. Unlike Hypertext Markup Language (HTML), XML allows users to define their own tags. These tags may also describe content semantics and are not simply limited to formatting as in HTML (Harold, 99). XML provides a universal meta-markup language that can be used to describe data, including configuration data used in a program. An XML document can be parsed at runtime and

the contents loaded into the running process. The data contained within the document can be as routine as stored configuration parameters, or more complex such as descriptions of components to be created and included in the current process, or even scripting code written using independent computer languages.

E. TEAPOT WORLD

Teapot World was created as a simple application in which to test the foundation protocols of NPSNET-V. The foundation protocols are five lightweight protocols that contain entity state information of varying complexities such as position, angular acceleration, and animation sequence start times. The foundation protocols are included in NPSNET-V and a developer can use them when creating their entities, instead of having to create unique protocols. Teapot World also allows testing of the DBP protocols, which have the same functionality as the foundation protocols.

The Teapot World application runs on the NPSNET-V architecture enabling manipulation of teapots in a Java 3D (Sun, 01) window. The model is an object, which inherits from the NPSNET-V entity class and has a robust set of attributes organized into lightweight Protocol Data Units (PDUs) for use in a 3-dimensional (3D) worldspace. The teapot view is imported from the Virtual Reality Modeling Language (VRML) into Java 3D. The controller is user input from both the keyboard and mouse.

Teapot World provides a baseline virtual environment that is suitable for testing protocols. Teapot World testing is performed both with two or more applications on the same machine and with applications running between two or more machines. By manipulating the teapot in the process in which it was created (also known as the master entity) with the keyboard and mouse, the copy viewed in another process (also known as

the ghost view) can be examined visually for proper behavior. The ghost copy can be compared directly with the master entity to check for the same alignment and rotation speeds as well as checking whether the animations run smoothly on the ghost side.

Because Teapot World is a simple environment, consisting only of one teapot, it is perfect for testing protocols since other variables that are present in a more complicated environment can be eliminated. An even simpler version of Teapot World used for network testing has no view component to the model, so that no processor time is needed for rendering graphics. This minimalist version enables further isolating the networking aspects of NPSNET-V using the Empty Entity, and allows sending packets at a rate which tests the limits of network capacity.

F. RELATED WORK - AREA OF INTEREST MANAGEMENT (AOIM)

The NPSNET-V AOIM scheme is the thesis topic of Navy Lieutenant Michael S. Wathen (Wathen, 01). NPSNET-V uses Area of Interest Management (AOIM) to structure network traffic. A major goal of the AOIM manager is to limit network traffic to only entities of interest to the application. Entities that are grouped together on the same geographic area, and receive each other's messages are considered to be in the same zone. These zones are based on geographic proximity and the number of entities that can share information based on system and network load. As the population in an environment increases, the number of zones increases and vice versa. A procedure whereby the zones split into smaller zones and the components are redistributed into these new zones is known as subdivision. The NPSNET-V AOIM procedures are dynamic, and the AOIM zones can be modified at runtime.

Dividing and combining AOIM zones is costly due to computation cost and communications delays, and so may not always be an ideal solution to optimize the number of participants in a region. Without information other than the number of entities in a zone, the AOIM manager is only able to make zone modification decisions based on simple high and low thresholds for the number of participants in a networked virtual environment. Further feedback is needed for effective large-scale optimization.

A network-monitoring implementation can provide network-state information to the AOIM manager. If a high number of entities are in the same zone, but are not generating many networking packets, the AOIM manager is now informed of this fact and may decide to not subdivide the zone. Conversely, on the other end of the spectrum, if a smaller number of entities are in a zone but the network is being overwhelmed, the AOIM manager can subdivide the zone to reduce the network load. The AOIM manager can look at the load on each channel as well, to make smarter decisions of how to distribute the entities into zones in a virtual environment. This process happens dynamically at runtime. AOIM is a continuous task since users are anticipated to constantly join and exit the environment, and also expected to cross zone boundaries frequently.

With a DBP network implementation, the AOIM manager has a further option in optimizing the distribution of entities in zones. DBP packets can have their packet resolutions changed rapidly, and at runtime. The network monitoring implementation informs the AOIM manager of the payload resolution of network packets, currently implemented as high or low. The AOIM manager may then choose to reduce or increase resolution of state information in these packets. A reduction in resolution reduces the

size of the PDUs, reducing the bandwidth used by the application and thereby providing a network protocol solution.

G. VIRTUAL REALITY TRANSFER PROTOCOL (VRTP)

Virtual Reality Transfer Protocol (VRTP) is a networking protocol research project intended to extend http and provide support for Large Scale Virtual Environments (LSVEs). LSVEs are primarily constructed as either peer-to-peer or server-based. VRTP provides a solution that enables both type of architectural designs, or even a hybrid approach, to be implemented. VRTP can serve as the sole protocol needed to support a LSVE. VRTP will combine many existing dissimilar protocols to provide this solution (Brutzman, 97). Network monitoring, such as the techniques presented in this thesis, is expected to be as important as client-server and peer-to-peer support.

1. Breakthroughs and Bottlenecks

Virtual environments continuously push the limit for computational complexity and networking availability. As technology continues to improve in these areas and remove bottlenecks, virtual environment design makes a leap forward until it hits the next bottleneck. VRTP implementations are expected to provide the breakthroughs needed to overcome the networking protocol bottleneck (Brutzman, 77).

2. Real-time Transport Protocol

Real-time Transport Protocol (RTP) supports applications where speed is of high importance. RTP adds quality of service and synchronization to existing transport protocols. RTP is ideal to support UDP-based applications such as streaming audio and video. Since speed is of vital importance in LSVEs, RTP will further contribute to the networking protocol efforts in an LSVE (Afonso, 99).

3. NPSNET-V

NPS has advocated and supported persistent online virtual worlds for years. A large-scale, persistent virtual world must have runtime-extensibility, scalability, and composability. One of the main challenges to achieving these goals is networking issues. NPSNET-V and DBP wil continue to develop as VRTP components. Together these provide a new protocol at the application layer to accomplish a variety of interdependent objectives and overcome diverse networking bottlenecks (Brutzman, 97).

H. SUMMARY

This chapter provides an overview of design patterns, basic network testing procedures, XML, and related work on the NPSNET-V project.

Using existing design patterns not only ensures that a good software-design approach is followed, but also provides a common terminology with which to discuss design and implementation issues. The network-testing considerations presented in this chapter provides a checklist with which to evaluate a thorough network-testing design. XML is a new and exciting technology that makes it possible to implement new solutions to many challenges, such as runtime-modifiable protocol specifications. Teapot World is a simple environment in which to measure metrics and test networking thresholds, while limiting interference from many of the other dynamic variables in a complex virtual environment. Concurrent and related work, especially in AOIM, significantly impacts on network optimization, since AOIM controls the channel allocations and is concurrently attempting runtime optimization. VRTP is a framework for integrating the variety of client, server, peer-to-peer and network monitoring protocols needed to provide essential network serveices for LSVEs.

THIS PAGE INTENTIONALLY LEFT BLANK

III. NPSNET-V

A. INTRODUCTION

This chapter provides an overview of the elements of NPSNET-V that affect both component protocols and network optimization. The key components affecting the relationships between entities and protocols are presented, along with flow of control of the network monitoring and optimization algorithm. Protocols included with NPSNET-V are specifically discussed. The goal to optimize NPSNET-V networking may differ somewhat from general network optimization, so this chapter provides additional background that illustrates how design strategies were derived and implemented.

B. MODEL / VIEW / CONTROLLER

The Model / View / Controller (MVC) pattern is commonly used in object-oriented programming for virtual environments. The *Model* object describes how an object interacts with its environment, containing the state variables and physics model of the object. The model contains critical attributes and computational procedures such as entity's position, velocity, and collision-avoidance algorithms. Most networking protocols in NPSNET-V directly implement the model component.

The *View* object provides the one or more visual depictions of an entity. One useful reason for having multiple-view objects is to enable multiple resolutions of an entity, or even multiple graphics formats. NPSNET-V currently supports both Java 3D and VRML/X3D component views.

The *Controller* object enables a consistent way to change values in the model object. The *Controller* object can accept input to change the model from many sources including user input, network traffic and multi-agent input.

C. ENTITIES

1. Entity Masters

Entity masters inherit from the entity class and describe the authoritative instance of an entity. Typically there exists only one entity master on a machine controlled by a user. In addition, there may be one or more entity masters, known as agent masters, controlled by an autonomous agent. All other instances of the same entity are a ghost copy of that entity. Entity masters include the singleton model class and typically also link a controller and a view.

2. Entity Ghosts

Every other instance of an entity that is not the master is an entity ghost. Often each host in a distributed exercise will contain one or more masters, which interact with ghost copies of other entities from other machines and any agent masters located on that machine. The ghost copies are local copies of remote masters and thus their state information may not necessarily be correct. Messages from the entity masters are sent to ghosts, who interpret the messages to update and extrapolate state values in the local copies of the entity's ghosts.

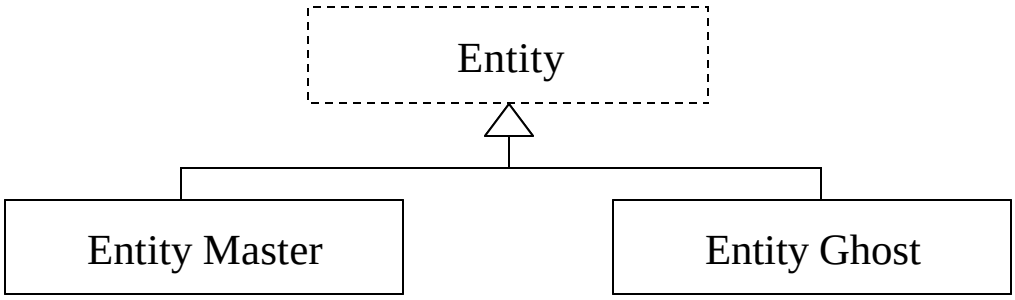

Figure 3. Inheritance hierarchies for abstract Entity interface and instantiable Entity Master and Entity Ghost interfaces

3. LDAP Server

All NPSNET-V entity class definitions exist on a web server. If a user wishes to create a new class of entity, they must either host a web server or arrange to have the class definitions placed on an accessible web server.

When a virtual world attempts to load a new entity, it doesn't necessarily have the class definition or know which web server it is stored on. When an unknown entity is encountered for the first time, a well-known Lightweight Directory Access Protocol (LDAP) server is queried, which provides a URL to the server that stores the entity class information. An LDAP server is a lightweight server that is similar in function to a DNS server. An LDAP server maintains a table of addresses of where to find information. In NPSNET-V, the LDAP servers maintain addresses of where to find copies of code (McGregor, 01). Replicated LDAP servers can cache replicated copies of such code and content, which thereby increases reliability, accessibility and scalability.

In the long term, VRTP-capable hosts participating in LSVEs are expected to provide individual serving capabilities.

4. Networking Overview

Entities in the NPSNET-V virtual environment need a means of communicating with each other. They accomplish this many-to-many message passing through application-specific protocols. Each entity has one or more protocols that report updates to the state information of that entity. The master instance of the entity understands how to generate Protocol Data Units (PDUs) describing changes in entity state. The ghost instances of this entity receive such messages and understands how to decode the PDU to make the appropriate, corresponding changes to the ghost.

The network flow begins when the controller object is notified of a change to the entity's internal state. The protocol generates a PDU with the appropriate data and passes this PDU to the Area of Interest Manager (AOIM). The AOIM decides where to send the PDU. The Entity Dispatcher is a singleton packet handler that directs all traffic flow of PDUs coming into and leaving the NPSNET-V application. The PDU message is then passed to the appropriate Channel object where it is written to the appropriate multicast (or unicast) network address/port combination corresponding to that AOI area of interest.

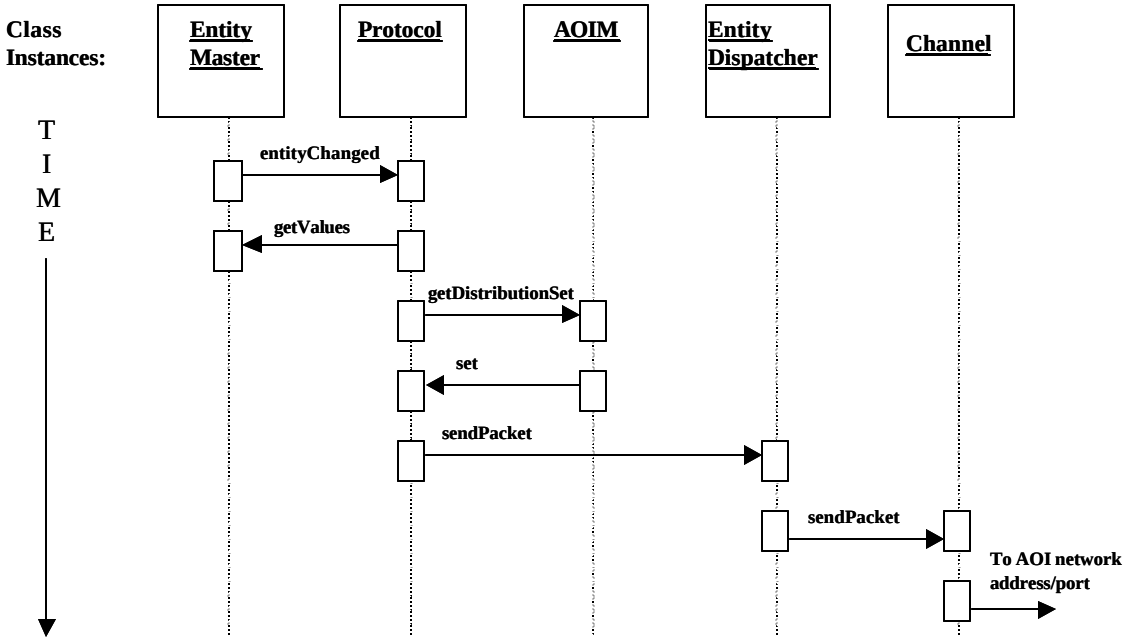

Figure 4. Event-sequence diagram for sending a network packet.

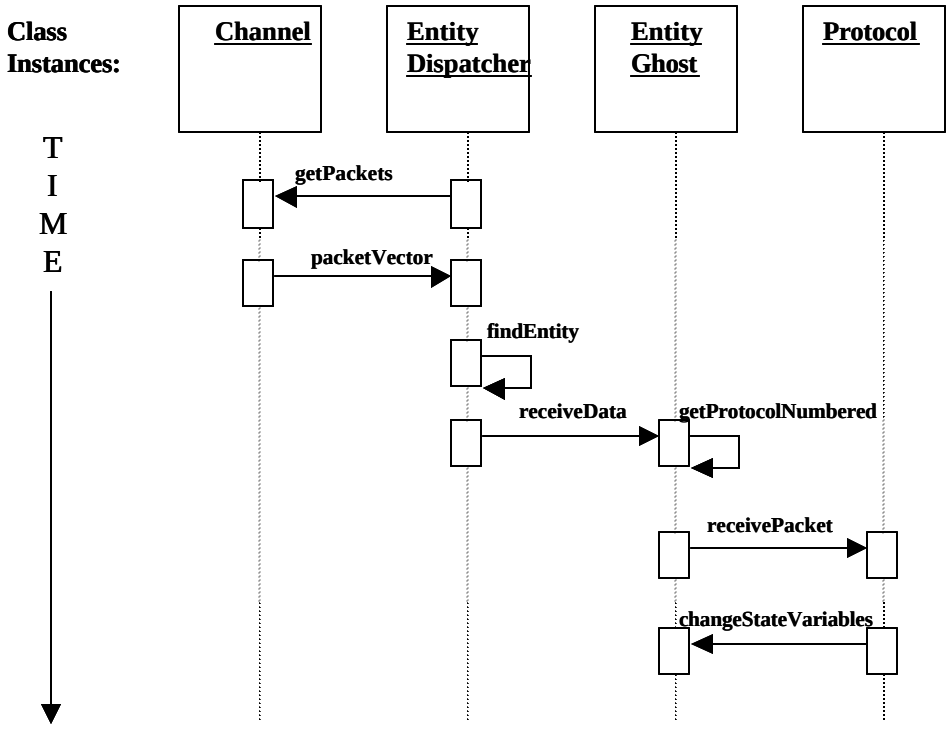

Figure 5. Event-sequence diagram for receiving a network packet.

The packet is received on channels of remote machines that are listening to traffic from that entity. The Entity Dispatcher inspects the payload header to determine the ghost entity that the message is intended for, and then forwards it to the entity. The ghost entity then decodes the message using the appropriate protocol, and applies the changes plus extrapolations to the ghost entity's state.

D. DYNAMIC BEHAVIOR PROTOCOL (DBP)

Java applications routinely use object serialization for networking communications. Unfortunately, this Java-specific approach requires the network

packets to be in very specific (and perhaps non-portable) binary formats. Using object serialization, the objects being passed and restored must utilize identical serialization in all respects. Sometimes conflicts can arise from simple, subtle differences such as serializing two identical objects using different compilers or different versions. Additionally, using serialized packet formats detracts from the interoperability of NPSNET-V, since these cannot be made to match existing non-Java packet formats.

Dynamic Behavior Protocol (DBP) uses Extensible Markup Language (XML) to describe packet formats and fields. Using DBP, we can create packets with specific layouts without relying on Java-specific object serialization. DBP protocols use a binary format on the network and can be created and modified at runtime. Once an XML protocol created this way has been distributed and parsed, a local entity knows how to read and write the PDU-packet format.

E. ENTITY DISPATCHER

The entity dispatcher is a singleton in NPSNET-V and is the central hub for message routing on each host machine. The entity dispatcher maintains a list of all entities in the system and controls the channel manager, which in turn controls each of the network channels used to send and receive network traffic. As shown in figure 5, incoming messages are passed from the channels to the entity dispatcher. The entity dispatcher determines which entity is addressed by the message and then forwards it. Outgoing messages follow the same flow in reverse.

F. PROTOCOLS

1. Protocols in NPSNET-V

Entities register their protocols with the entity dispatcher. Each protocol is assigned a unique Globally Unique Identifier (GUID) for improved performance. When

a message encoded using an unknown protocol is received, the entity dynamically loads the protocol component associated with that GUID, and packet payloads corresponding to that protocol are thereafter understood by the owning entity.

2. NPSNET-V Foundation Protocols

Five foundation protocols are provided by the NPSNET-V architecture. They are the transform, inertial, animation, articulation and acceleration protocols (McGregor, 01). Each of the five protocols inherits from the protocol base class. The transform protocol provides spatial position, orientation and scale operations. The inertial protocol conveys linear and angular velocity. The animation protocol allows simple animations, the articulation protocol passes data for articulation of additional joints, and the acceleration protocol adds data for linear and angular acceleration.

The five foundation protocols provide a robust, lightweight network-communicaitons package to communicate entity state changes. Since the protocols are included in NPSNET-V, any application running on top of NPSNET-V can use these protocols for its entities. In the event that these protocols do not meet the requirements for an application, additional specialized protocols can be loaded dynamically.

3. DBP Protocols

The DBP protocols are a separate protocol package included with NPSNET-V. The DBP protocols include examples that are identical to the foundation protocols in functionality, each having the same names and field descriptions.

DBP protocols are defined by one or more XML documents that describe the layout of the protocols. Even though the field descriptions are the same as for the foundation protocols, the data types may be different in order to experiment with

performance optimization. As an example, a foundation protocol field defined as having type double may be described in the corresponding DBP protocol as a float. Once a protocol-defining XML document has been parsed, the protocol is ready to send and receive packets without having to parse the definition document again later. Different versions of a protocol can be described by different XML documents, and thus a protocol can be redefined, communicated, reparsed and thus updated at runtime.

G. NETWORK CHANNELS

1. Channels

NPSNET-V establishes AOIM-organized channels, each of which encapsulates a socket for network communications. The AOIM determines which channel a protocol packet is sent on. Typically, all entities within the same AOIM zone have their protocols send updates on the same channel, corresponding to a particular geographic area. Limiting network traffic within a zone to a multicast channel enables individual host machines to disregard packets that are not of interest to active entities on the local machine. This culling of unwanted network packets via multicast-hardware capabilities improves performance and enables effective scalability to much higher aggregate traffic levels.

2. Channel Manager

The channel manager is a singleton created by the entity dispatcher. The channel manager keeps track of channels that have been created.

H. SUMMARY

This chapter provided an overview of the elements of NPSNET-V that affect protocol design and network optimization. NPSNET-V follows the Model / View / Controller (MVC) paradigm, which enhances the extensibility of the design architecture.

The entities in a virtual environment each have two main components: the master entity resides on the owning machine and maintains the authoritative state information, while the ghost copies reside on every other machine that has knowledge of the entity. The DBP protocols use XML documents that are parsed to define the data types and initial values. This flexibility can be exploited to enable different resolutions for each protocol and dynamic run-time updates. The AOIM scheme uses channels to divide the networking traffic among different multicast (or unicast) address/port combinations for hardware culling of unwanted network packets and scalable traffic handling.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. DESIGN OF EXPERIMENTS

A. INTRODUCTION

This chapter discusses the experimental approaches used to demonstrate and evaluate the main goals of this thesis. There are three separate efforts. The first effort is creating XML-based protocols. The second is constructing a network monitor. The third effort is determining the networking points of failure that can occur while running a virtual environment, and thus establish default thresholds for protocol switching.

B. DBP PROTOCOLS

The basic protocols already included with NPSNET-V are the five foundation protocols. These implemented protocols make an excellent starting point for implementation via the DBP protocols, thereby enabling experimental comparisons. Five DBP protocols were created based on the foundation protocols: transform, inertial, acceleration, animation, and articulation. Since these DBP protocols mirror the foundation protocols in functionality, they can be compared directly to verify functional correctness and also to test for any inherent advantages provided by DBP flexibility.

The entities that the DBP protocols are to interface with in NPSNET-V are written in Java. They have coded expectations for parameters that can be received and sent. For this reason, the DBP protocols each require that the supporting XML documents define these same parameters. Although this may limit the ability of the XML documents to describe parameter definitions for use in the protocols, the data types for these definitions can be changed to meet changing requirements.

To take advantage of the dynamic abilities of the DBP protocols, and to simply achieve useful implementations, each protocol is described with both a high-resolution

and a low-resolution XML document definition. To accomplish this, each value normally defined as a 64-bit double in the normal high-resolution version has a counterpart defined as a 32-bit float in the low-resolution version. Additionally, each 64-bit long data type in the high-resolution protocols has a corresponding 32-bit integer datatype definition in the low-resolution protocols. When a packet is sent, the appropriate data types are written to a byte stream based upon which protocol resolution the protocol has been directed to use. On the receiving end of a packet, the protocol tests each mutable value that arrives to see which data type is being used. This implementation allows great flexibility when specifying varying protocol resolutions.

To quantitatively test the correct behavior of the DBP protocols, a network monitor and a visual display were created. The network monitor counts the number of bytes sent and received, and the display creates a graph that shows this information. Two identical entities are created that use the same DBP protocols. The entities both send packets at the same rate, implemented with a heartbeat timer. They are compared at the same time, one with protocol set to high and the other to low resolution. Next, the same test is performed comparing the DBP protocols to their corresponding Foundation protocols.

C. NETWORK MONITOR

A network monitor was created to gather statistical data about network performance for use in optimization. The networking metrics selected for measurement are packets sent and received, and bytes sent and received. These two metrics allow measuring the data rate in packets-per-second, as well as the bandwidth used by an application. Data rate and bandwidth are two measurable indicators that can be used to

predict network load. Knowing this information can be used for determining whether to switch protocol resolutions at runtime and take advantage of the DBP protocol flexibility.

Whenever packets are sent or received through the entity dispatcher, the network monitor is notified of the number of bytes and number of packets sent or received. Following the observer pattern, an application interested in the network data instantiates a network monitor and queries it at the desired interval.

The data collection point selected is at the Entity Dispatcher. The Entity Dispatcher is a singleton, which ensures no double reporting of data, and all network traffic flows through it, which ensures that all packets are counted. The Network Monitor was initially designed as a singleton, but was later changed to a private data member of the singleton Entity Dispatcher. A redundant data collection system has been implemented with data collection for every channel, but this is currently disabled and not needed. Data collection in the Entity Dispatcher can be turned on or off at the discretion of the programmer.

The Network Monitor collects data for every packet sent or received. Summary data is returned to those subscribed listeners that request it. The returned network-monitoring information contains eight values:

- average number of packets received
- average number of bytes received
- average number of packets sent
- average number of bytes sent
- number of packets received since the last report

- number of bytes received since the last report
- number of packets sent since the last report
- number of bytes sent since the last report

The Network Monitor has an adjustable time duration for data-collection buffers. The averages computed are averaged over this time interval. Data is dropped from the buffer list when older than the specified duration. The resulting averaging process is similar to the sliding-window procedure used in TCP connections (Peterson, 00).

The Network Monitor reports the status of the network as quiet, busy, or thrashing when queried. The thresholds for when the network is considered busy or quiet can be set through function calls, and are initialized to default values based on the network-testing portion of this thesis. If the measured network state repeatedly crosses back and forth across the high and low thresholds, then the network state can be considered thrashing. Thresholds for the number of times the state attempts to switch and the time interval across which these attempts occur can both be set through the interface to the network monitor.

D. EVALUATION STUDY

Following the steps presented in Chapter II, the experiment follows the steps presented in "The Art of Computer Systems Performance Analysis" (Jain, 91):

1. The goals of the experiments are:
    - Determine the correlation between packets-per-second and CPU utilization
    - Determine bandwidth utilization

- Determine packets-per-second at which CPU usage is at 10%
- Find ways in which packets can be modified in order to optimize network performance

Since a machine typically receives many more packets than it sends, the point of measurement is at the receiving machine.

2. For testing, only one machine is the sender, and the other the receiver. The sender produces packets of a fixed size at a steady rate. The receiver consumes and measures the sent packets. The two machines are disconnected from any other networks to eliminate any interference or noise. Both sender and receiver use the Empty Protocol, which passes a fixed-size byte-payload.

3. The metrics selected are discussed in paragraph F below and include bandwidth, packets-per-second, and CPU utilization.

4. The system parameters are discussed in paragraph G below. Network benchmark testing is performed between two machines connected by either a 100 Mbps Ethernet connection or a 56K modem connection. This testing is performed using likely packet sizes at varying packet rates in 10-second transmission windows.

5. The key parameters for the experiment are listed in Figure 6 and defined as follows:

- The available bandwidth is set at two different levels (Ethernet 100 Mbps and 56K modem).

- The packet size is set at three different levels, representing the transform high, transform low and animation high packet definitions.
- The packets-per-second sending rate is varied at regular intervals.

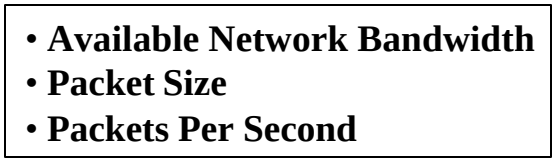

Figure 6. Modifiable Experiment Parameters

6. The evaluation techniques are discussed in paragraph E below and include both simulation and measurement-based network-evaluation techniques.

7. The workload consists of an empty entity that transmits byte arrays of the three protocol packet sizes at an adjustable rate.

8. For the experimental design, packets are sent at steady rates, over a range of flow rates. Three iterations are performed for measurements at each transmission rate.

9. For the data analysis, the data is examined to determine if conclusions can be drawn from the samples taken.

10. For the display presentation, the results are plotted to determine the optimal and maximum desired packet rates, and then metrics are compared against each other to determine if any correlation exists among the results.

E. EVALUATION TECHNIQUES

1. Simulation

The simulation conducted is performed with an empty entity that transmits packets of three tested sizes. The first size, 138 bytes, represents a high-resolution transform protocol packet. The second size, 74 bytes, represents a low-resolution transform packet. The third size, 19 bytes, represents a high-resolution animation protocol packet. This simulation determines how well the receiving machine and transmission media might handle different packet flow rates for these different packet sizes. The percent packet loss and percent CPU utilization are measured at the receiving machine.

2. Measurement

The measurement is performed after the results have been analyzed and benchmarks are established. The benchmarks are established based on observations regarding how well the networking optimization performs during an actual simulation.

F. METRICS

1. General

In conducting experiments to determine the appropriate settings for the DBP protocols, three metrics (listed in Figure 8) are used: bandwidth used, packets per second, and CPU usage. The experiments seek to quantify the relationship between these three metrics and the proper settings for the DBP packets to achieve application-based optimization.

Figure 7. Experiment Metrics

2. Bandwidth

The definition of bandwidth used in this thesis is the aggregate bits-per-second provided to the application. The term available bandwidth refers to the bandwidth provided by the physical media used to connect the network. Changing the type of connection between two machines can change the maximum bandwidth available. The two tested connections are an Ethernet 100 Mbps connection and a 56K bps modem connection. These two connections represent the target 100 Mbps high end and 56k bps low end of anticipated client connections in web-based distributed virtual environments.

3. Packets per Second

The packets per second metric simply measures the number of packets sent over a one-second interval. Packets are measured at that rate for a four-minute time period and the average packets-per-second value calculated. The four-minute measurement window is selected for two reasons. First, a sufficiently long duration is needed to simulate realistic application conditions. Second, a surge in CPU utilization occurs approximately every 3 1/2 minutes, which is correllatable and attributed to Java garbage collection of transient data structures. Since data collection must occur under realistic conditions, the measurement time interval must not eliminate or avoid this occurrence. The four-minute data-capture window ensures consistent inclusion of a single garbage-collection event.

The high-resolution transform protocol payload is 16 doubles, which equates to 128 bytes. The high-resolution inertial protocol has a payload of 13 doubles and 2 longs, for a payload size of 120 bytes. The high-resolution animation protocol has a payload of 1 long and 1 byte for a payload size of 9 bytes. The high-resolution acceleration protocol has a payload of 6 doubles for a payload size of 48 bytes. The high-resolution articulation protocol has a payload of 4 doubles for a payload size of 32 bytes. Each protocol has an internal overhead of 10 bytes and NPSNET-V prepends a header of 24 bytes.

The low-resolution transform protocol payload is 16 floats, which equates to 64 bytes. The low-resolution inertial protocol has a payload of 13 floats and 2 integers, for a payload size of 60 bytes. The low-resolution animation protocol has a payload of 1 integer and 1 byte for a payload size of 5 bytes. The low-resolution acceleration protocol has a payload of 6 floats for a payload size of 24 bytes. The low-resolution articulation protocol has a payload of 4 doubles for a payload size of 16 bytes. Each protocol has an internal overhead of 10 bytes and NPSNET-V prepends a header of 24 bytes.

External to NPSNET-V, each packet has a 20 bytes header for IP, and a 8 byte header for UDP. Systems connected using Ethernet have an Ethernet header of 18 bytes prepended. Systems connected by modem using Point-to-Point Protocol (PPP), have a PPP header of six bytes, and the IP and UDP headers are reduced to a total of 3 bytes (Stevens, 94). The total overhead external to NPSNET-V using an Ethernet connection is 46 bytes, and using PPP on a modem is 9 bytes.

Protocol	Payload	Total Packet Size NPSNET-V	Total Size Ethernet	Total Size Modem
Transform High	128	162	208	173
Transform Low	64	98	144	109
Inertial High	120	154	200	165
Inertial Low	60	94	140	105
Animation High	9	43	89	54
Animation Low	5	39	85	50
Acceleration High	48	82	128	93
Acceleration Low	24	58	104	69
Articulation High	32	66	112	77
Articulation Low	16	50	96	61

Figure 8. DBP Packet Sizes

The most common type of packet in a networked virtual environment is a packet updating a participant's position, analogous to the DIS protocol ESPDU (Singhal, 99). The DBP protocol that provides analogous functionality is the transform protocol. The transform protocol is 98 bytes in size at low-resolution and 162 bytes at high-resolution. The transform protocol is also the largest-sized DBP packet type, so if conditions are tested using the transform protocol, packets from the other protocols will fit within the bandwidth footprint tested.

4. CPU Usage

A host CPU consumes processing power for each packet that it sends and receives. It is important to measure how much work the CPU must do to process network packets. Of further interest is seeing whether correlations exist between CPU usage and packets-per-second.

The amount of CPU processing used to process packets from the network needs be balanced with other requirements. Keeping the network's resource demand low

provides greater host resources for interaction calculations, graphics rendering, and other tasks. A design decision was made by the NPSNET-V Research Group to keep this processing to between 10% and 15% maximum CPU capacity. The target maximum packet-per-second transmission rate will thus correspond to the CPU usage at 10%.

G. TESTING

1. Determining Benchmarks

For the DBP protocol's ability to switch resolutions to be useful, such switching must occur automatically and at the appropriate times. One of the goals of testing the impact of packet rates on the receiving machine is to determine the optimal and maximum packet flow rates. Once benchmarks are established, the program can better collect data during execution and then make network-optimization decisions by comparing the collected data to these benchmarks.

2. Protocols

a. Empty Protocol

For testing, a new Foundation protocol was created, the empty protocol, which merely transmits an empty packet with a predetermined byte size. The entity master is on the sending machine and the ghost is on the receiving machine. The empty protocol is used to gather information about packets dropped and CPU usage at the receiving machine. This testing is done at a variety of packet rates, including sending as fast as possible from the sending machine. The sending rates are set based on spreading the rates out to achieve an even distribution between zero and the maximum sending rate. The rates are adjusted by adding intermittent delays to slow down the sending rate. This is done rather than sending a specified number of packets per second which would cause surging behavior instead of the desired even flow rate.

b. DBP Protocol

The DBP protocols have high and low resolution settings. Parsing an XML document and reading in the appropriate data types for each parameter of the protocol sets the resolutions. The high-resolution format uses data types that are synonymous with the foundation protocols. The low-resolution version of this protocol degrades 64-bit doubles to 32-bit floats, and 64-bit long integers to 32-bit integers.

To compare the effects of these savings in packet size, the number of bytes transmitted over a minute at the high resolution is compared to the corresponding number of bytes at low resolution. Additionally, these measurements are compared to measurements of the foundation protocols of the same type. To achieve synchronization and consistency, the protocols transmit periodic "heartbeat" packets, and data is averaged over a minute.

3. Test Environment

The platforms used for testing are two Pentium III 1-Ghz computers with 256 MB RAM. These systems use Windows 2000 with Service Pack 2 as their operating system. The programming language used is Java's JDK 1.3.1 (Java, 01). The sending platform runs an Apache web server (Apache, 01) and an LDAP server (Eudora, 01). The receiving machine runs the network monitor application to measure CPU usage.

When tests are run between the systems, the fast connection is a 100 Mbps Ethernet connection using multicast. The slower connection is between two external 56k modems running PPP connections and using UDP unicast packets. In each case, the machines are isolated on their own standalone network, with minimal applications running in the background.

H. PACKET RESOLUTIONS

1. DBP

The dynamic behavior protocols can have their resolution adjusted at runtime. Merely changing the XML documents that describe the protocol packets can create varying resolutions. Only the payload (and not the header) of the packet is reduced.

2. Other Protocols

To change the resolution of any protocol other than the DBP protocols, a programmer must create a new protocol, which enables the resolution changes in the protocol. This flexibility is only possible following extensive manual programming, compilation, testing and debugging by experienced programmers.

I. SUMMARY

This chapter discusses the design of experiments used to conduct all three major portions of this thesis. The DBP protocols are written in Java, and each have supporting XML files that describe the data type definitions. The network monitor reports the current state of the network to indicate when to switch resolutions with a DBP protocol. The network testing is designed to establish the optimization points in the NPSNET-V networking architecture, and identifies where to set the network monitor state thresholds. Extensive parameter analysis and metric values are discussed.

THIS PAGE INTENTIONALLY LEFT BLANK

V. EXPERIMENTAL RESULTS AND ANALYSIS

A. INTRODUCTION

This chapter presents the results of the experiments corresponding to the three main objectives of this thesis: DBP Protocol implementation, the network monitor, and the analysis of protocol changes for network optimization. The majority of this chapter is devoted to conducting isolated network testing to quantify the critical break points in CPU utilization and percent packet loss. Packets-per-second and CPU utilization graphs are presented with plotted means and standard deviations.

B. DBP PROTOCOLS

1. Protocol Switching

Two identical entities are compared to each other, one using high and the other low-resolution DBP protocols. These two entities both used the transform protocol. The only means of sending packets is through a heartbeat thread, which sent a packet of each type, by each entity, once per second. The NPSNET-V system network-status utility was used to compare the two entities to give a graphic representation of the byte savings when switching resolutions. The low-resolution only transmitted approximately 60% of the bytes that the high-resolution protocol did.

Comparing the DBP transform protocol directly to the hard coded foundation-version transform protocol, shows much greater savings in bytes sent. The high-resolution DBP transform protocol sends approximately 25% of the number of bytes as its foundation counterpart. The low-resolution DBP transform protocol sends only 15% of the bytes compared to the foundation protocol.

Reducing packet sizes for Ethernet connections does not yield a significant advantage. Reducing the number of packets sent, however, does result in a performance gain. This performance gain primarily comes from reduced processing requirements in the variety of threaded Channel, Entity Dispatcher, Entity Master / Ghost, and Protocol classes that handle the packet distribution. Further performance gains can be expected for LSVEs distributed across Wide-Area Networks (WANs) due to reduced packet-routing overhead in routers. WAN optimization for widely distributed LSVEs will be an important area for future work.

When high numbers of packets are being sent using either different protocols or from different entities, we can gain efficiency by combining some of the data into packets. Since the data sent in a virtual environment is timely, using such a strategy must have an either / or condition for sending data, where either enough data exists to fill a packet, or a short timer has expired. If packet aggregation were incorporated into NPSNET-V, the savings result is 4 to 6 ½ times fewer packets sent whenever an entity is actively sending packets.

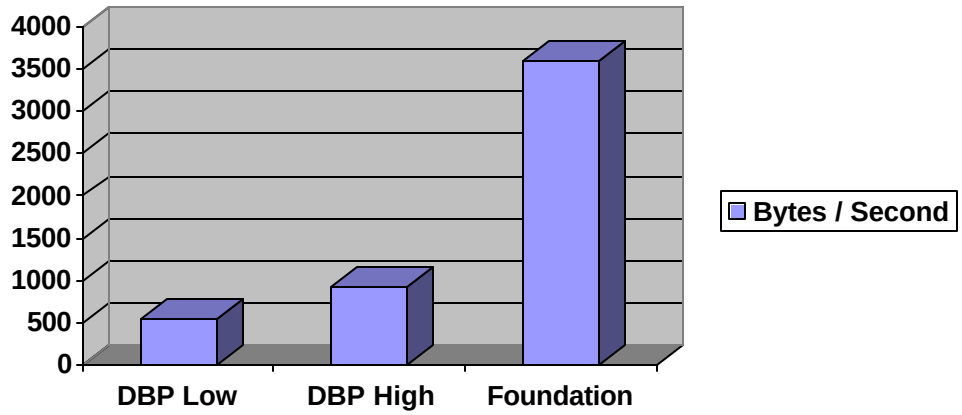

Figure 9. Comparison of Protocols' Bytes Per Second

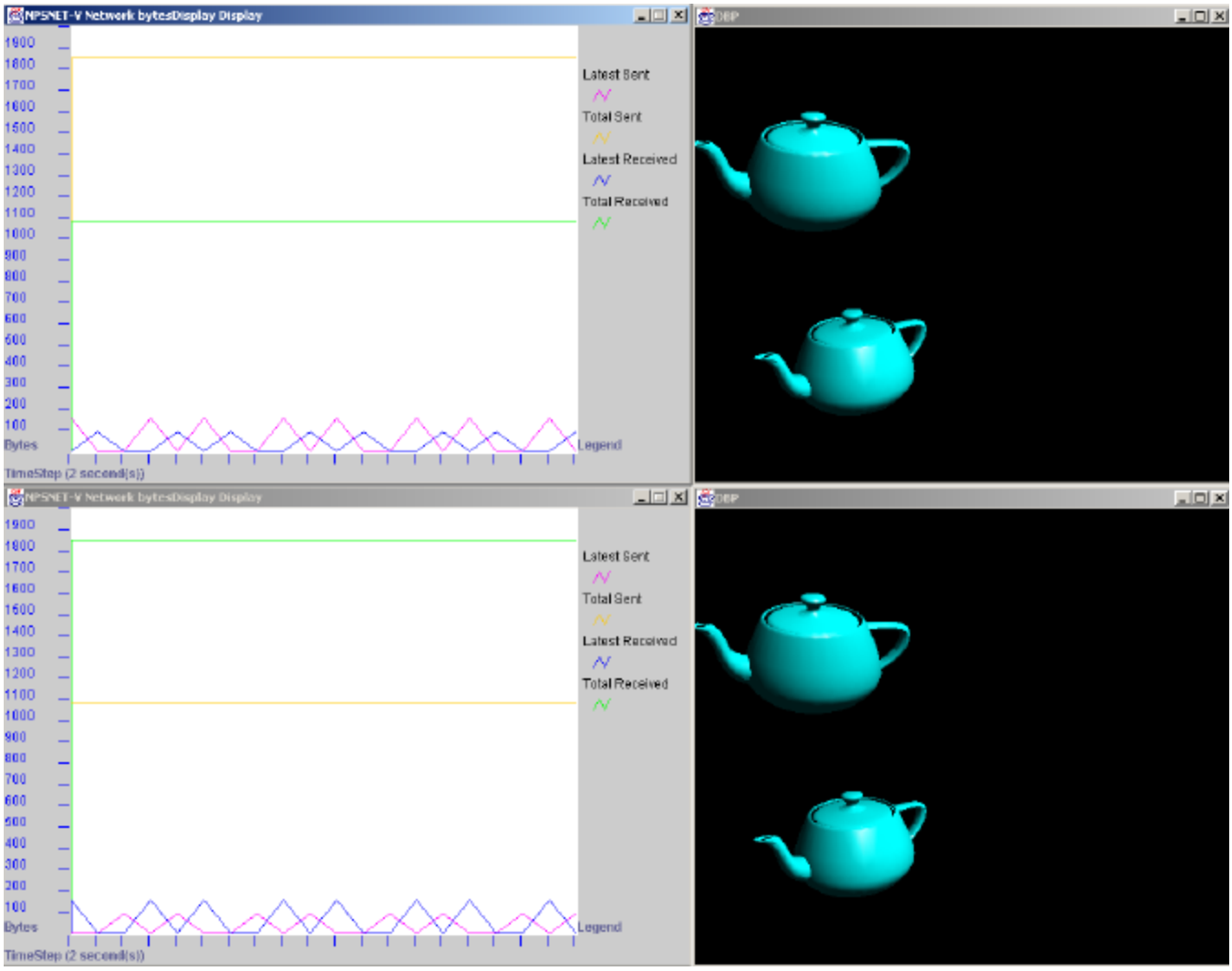

Figure 10. DBP Transform Protocol Running at Low and High Resolution

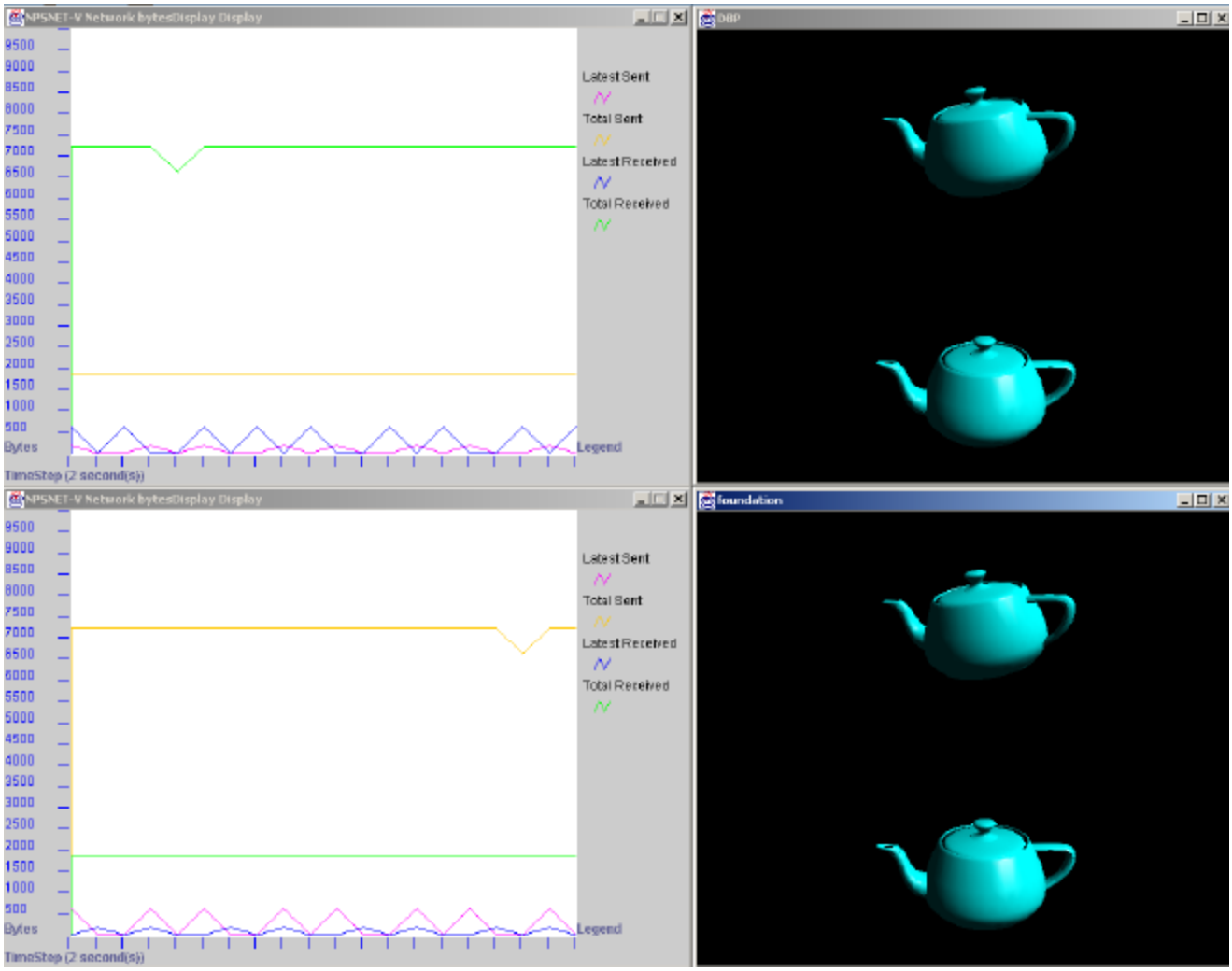

Figure 11. DBP and Foundation Transform Protocol Running at High Resolution

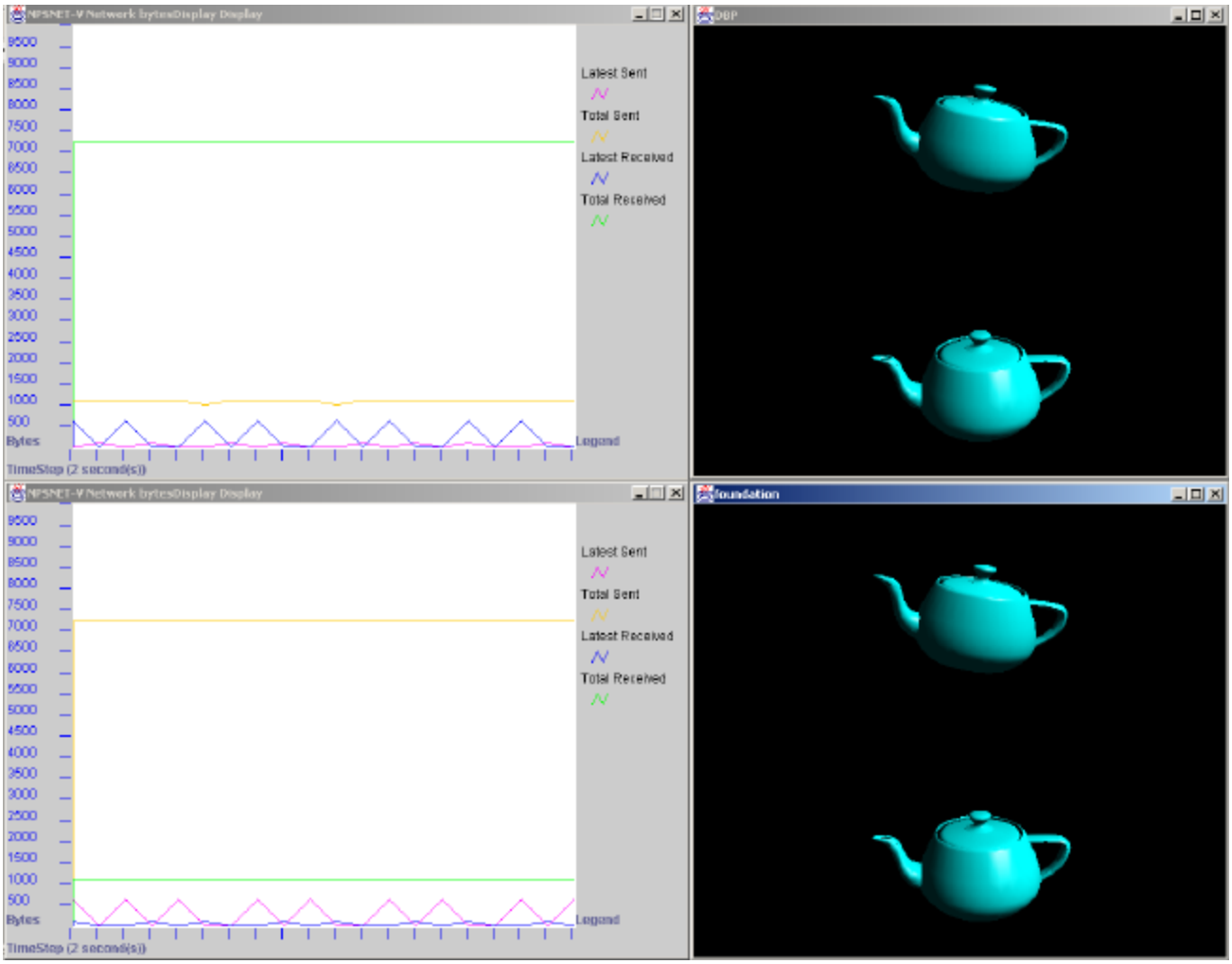

Figure 12. DBP and Foundation Transform Protocol Running at Low Resolution

2. XML Descriptions

The DBP protocols each have two pieces: a general-DBP Java program that describes the functionality of the protocol, and an XML document that describes the data types and initial values for the protocol. The relationship between the general-DBP Java class file and XML documents that define protocols is one-to-many. The first time each

XML document is used by a protocol, it must be parsed. During testing, the protocols parsed the required documents smoothly and without error. Each XML file can be edited with any text editor. The XML documents can have their data-type descriptions and initial values changed quickly and easily this way.

C. NETWORK MONITOR ANALYSIS

Operation of the network monitor has been verified by comparing known packet and byte quantities sent and received with the quantities reported by the network monitor. The output graph displayed by the monitor is easy to read and gives a clear picture of bothe current and ongoing network state.

A concern, as with most monitoring utilities, is whether the load put on the system to gather networking data is worth the information gathered. A series of empty protocols tests were run without any network monitoring actions being performed. The transform-high protocol was selected since it puts the greatest load on the system. The CPU utilization was reported and compared to the same test in which the data-gathering was enabled. These measurements are presented in figure 12. They show that approximately 2 ½ % of the CPU utilization is required to collect this data at the target 10% CPU Utilization.

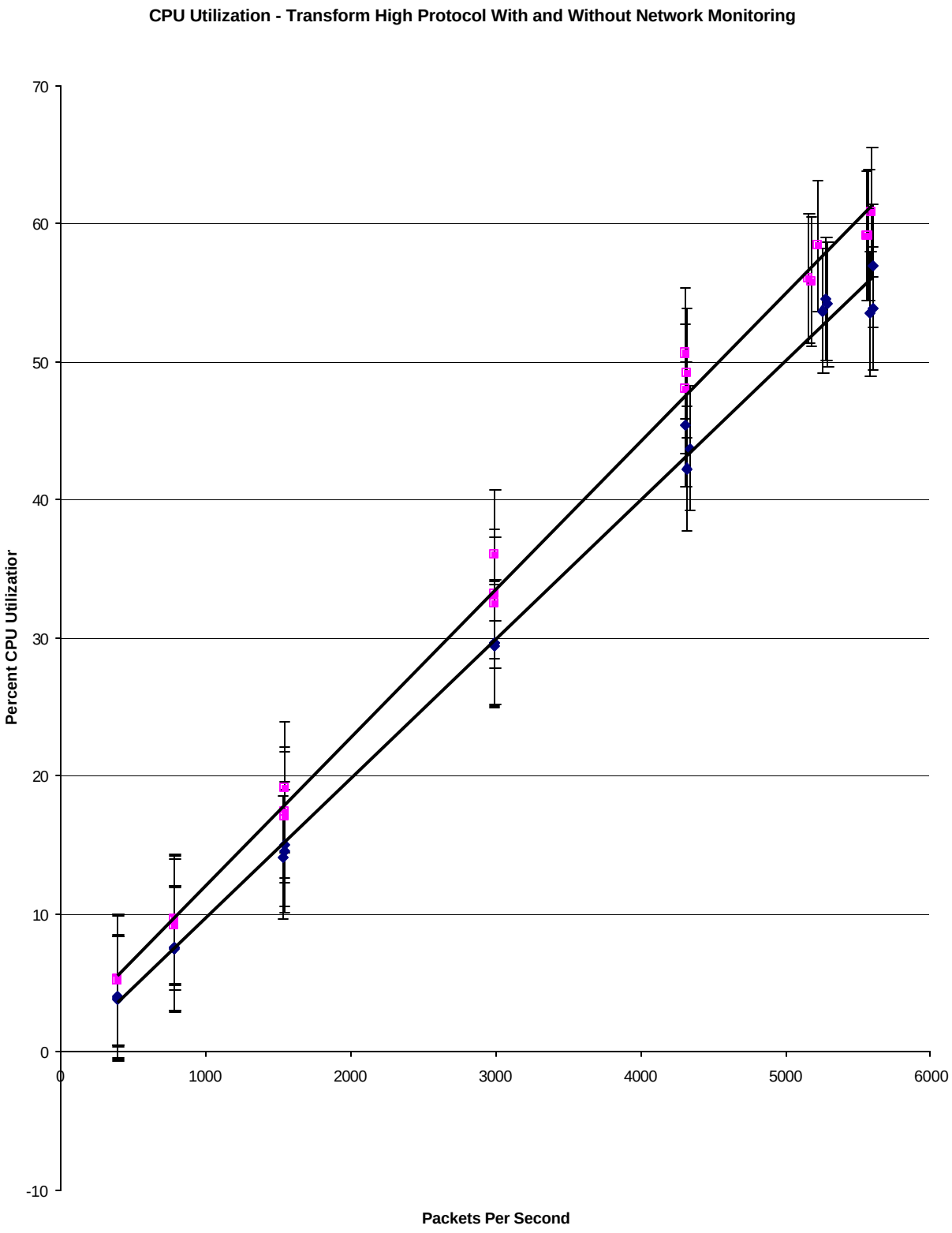

Figure 13. CPU Utilization Transform High - With and Without Data Collection

D. METRICS ANALYSIS

The metrics analysis investigates the effects on CPU load and packet loss of various packet sending rates, packet sizes, and physical transmission media. The results of this analysis are used to determine what, if any, relationships exist between these metrics.

1. Ethernet Packet Loss

The maximum rate that packets can be sent was minimally affected by the packet size. The maximum sending rate was approximately 5,600 packets-per-second. The packet loss for the Ethernet testing was affected mostly by the rate at which the packets were sent, and unaffected by the packet size. The packet loss for all three packet types increased exponentially with the sending rate. The best fit is a third-order polynomial graph. The packet losses for all packet types were between 1.5% and 4.5% at the maximum sending rate.

The packet loss for Ethernet testing was also affected by the size of the packets being sent. For both the high-resolution (138 bytes) and the low-resolution (74 bytes) transform packets, minimal (0.1%) packet loss began at a sending rate of 800 packets-per-second. For the high-resolution animation protocol packet, the same minimal packet loss began at about 1,500 packets-per-second. At the maximum sending rate, the loss rates were around 4%, 2.5% and 1.5% for the transform-high, transform-low, and animation-high protocol packets respectively.

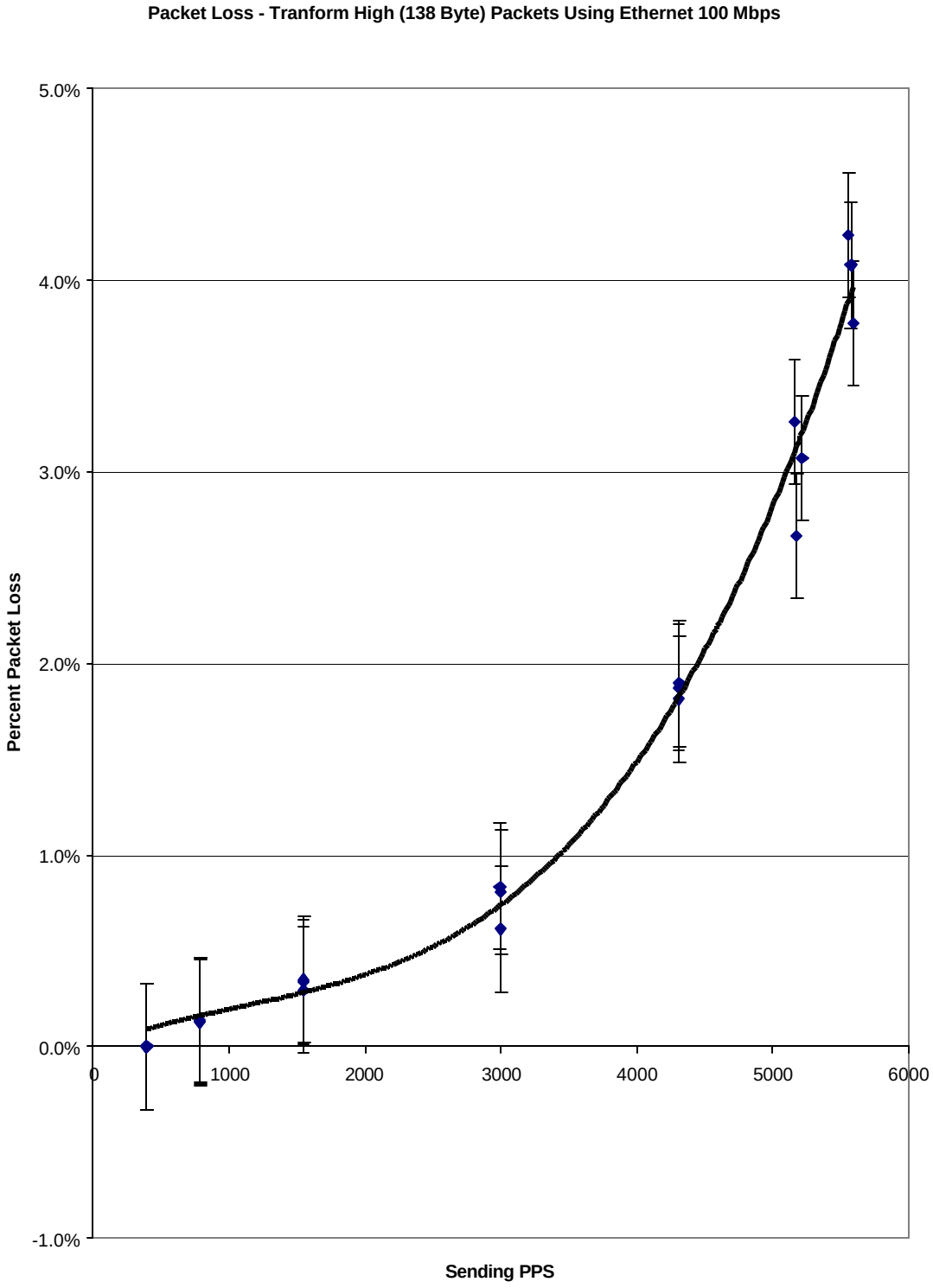

Figure 14. Packet Loss Transform High – Ethernet (maximum value 4.2%)

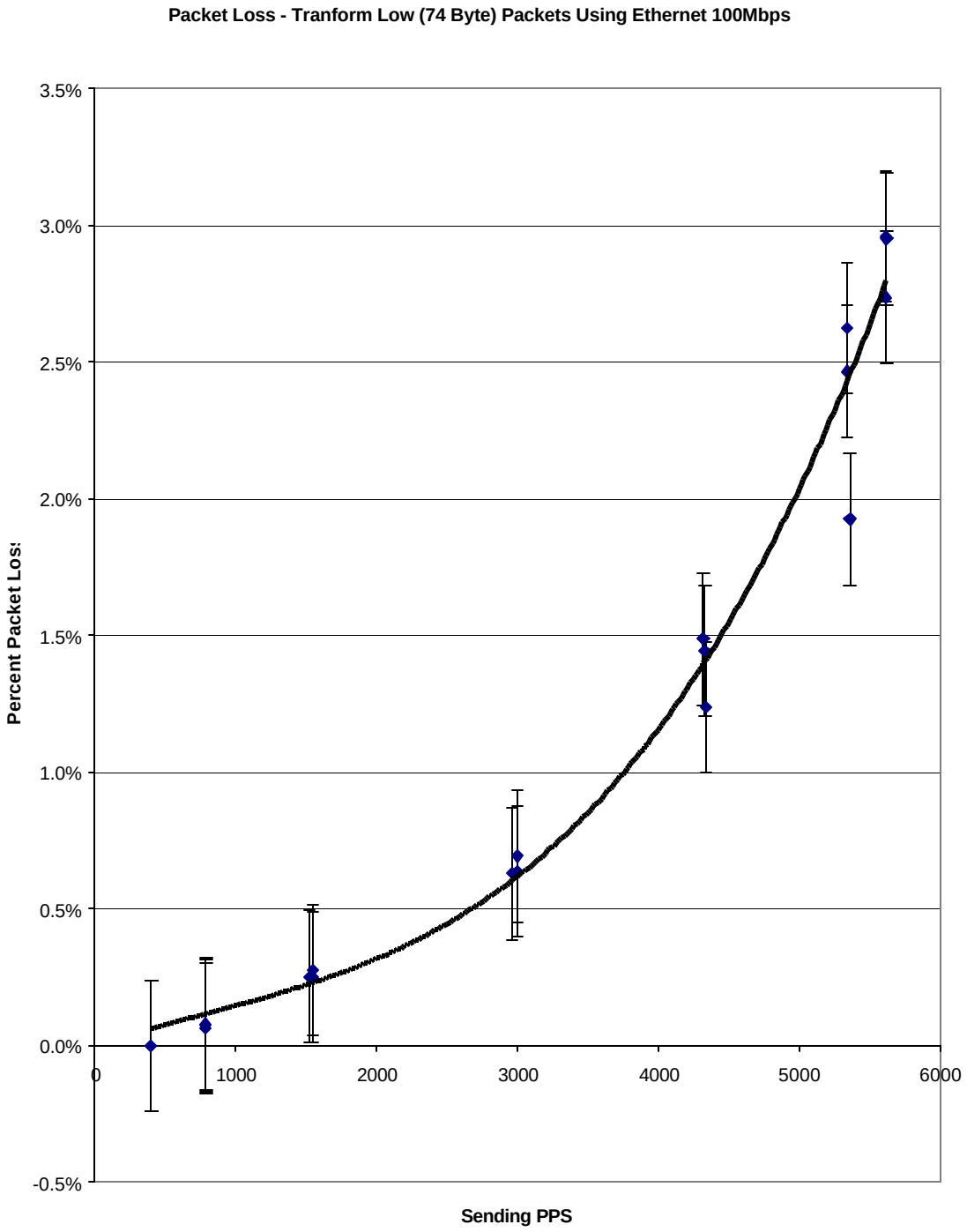

Figure 15. Packet Loss Transform Low - Ethernet (maximum value 2.9%)

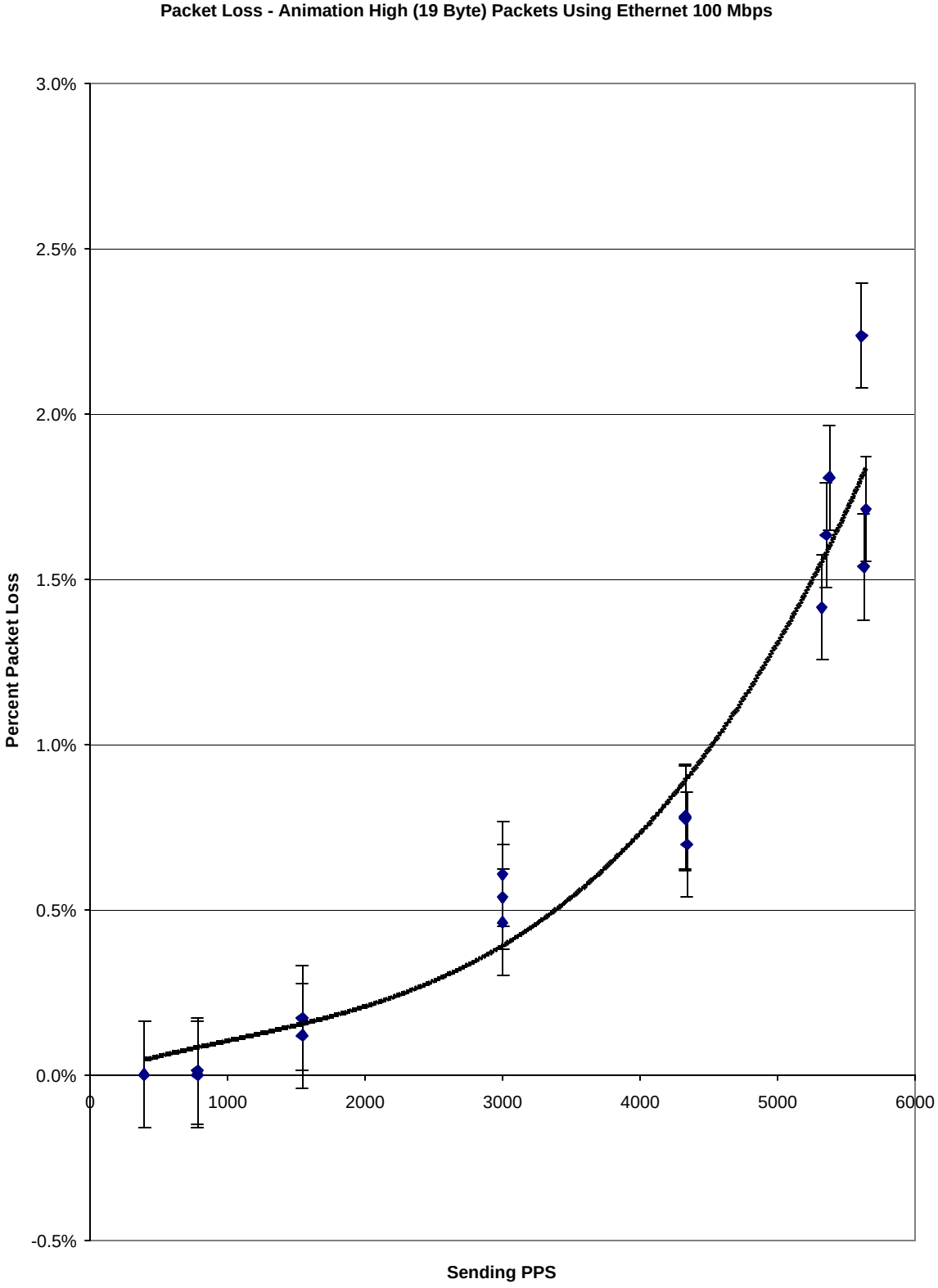

Figure 16. Packet Loss Animation High - Ethernet (maximum value 2.2%)

2. MODEM Packet Loss

For all three packet sizes and for various sending rates, including the maximum sending rate, there were no packets lost over the modem connection. The maximum achieved sending rate by modem was 94 packets-per-second, which is not sufficiently large to cause the receiving CPU to drop any packets.

The constrained available bandwidth did not cause the UDP packets to drop during the send operation as initially predicted. The TCP/IP stack checks the socket send buffer and if it is full, blocks the process from sending more packets. This happens in the `sosend()` method in the TCP/IP stack. The modem send-buffer fills quickly, since it is constrained by the slow connection, and imposes constant delays on the sending application. This happens even though the packets are UDP.

3. Ethernet CPU Usage

For Ethernet connections, the load on the receiving machine's CPU is linearly proportional to the sending rate as measured in packets-per-second. The CPU utilization percentage is the same for all three packet types sent. The maximum CPU utilization is approximately 60%, at the highest achieved sending rate of approximately 5,600 packets–per-second.

The lack of correlation between packet size and CPU utilization is not surprising. The CPU must use cycles to process each packet that arrives, which can add up quickly. This is one reason why Ethernet cards are generally set to listen to certain ports and not be promiscuous (i.e. not listen to all ports).

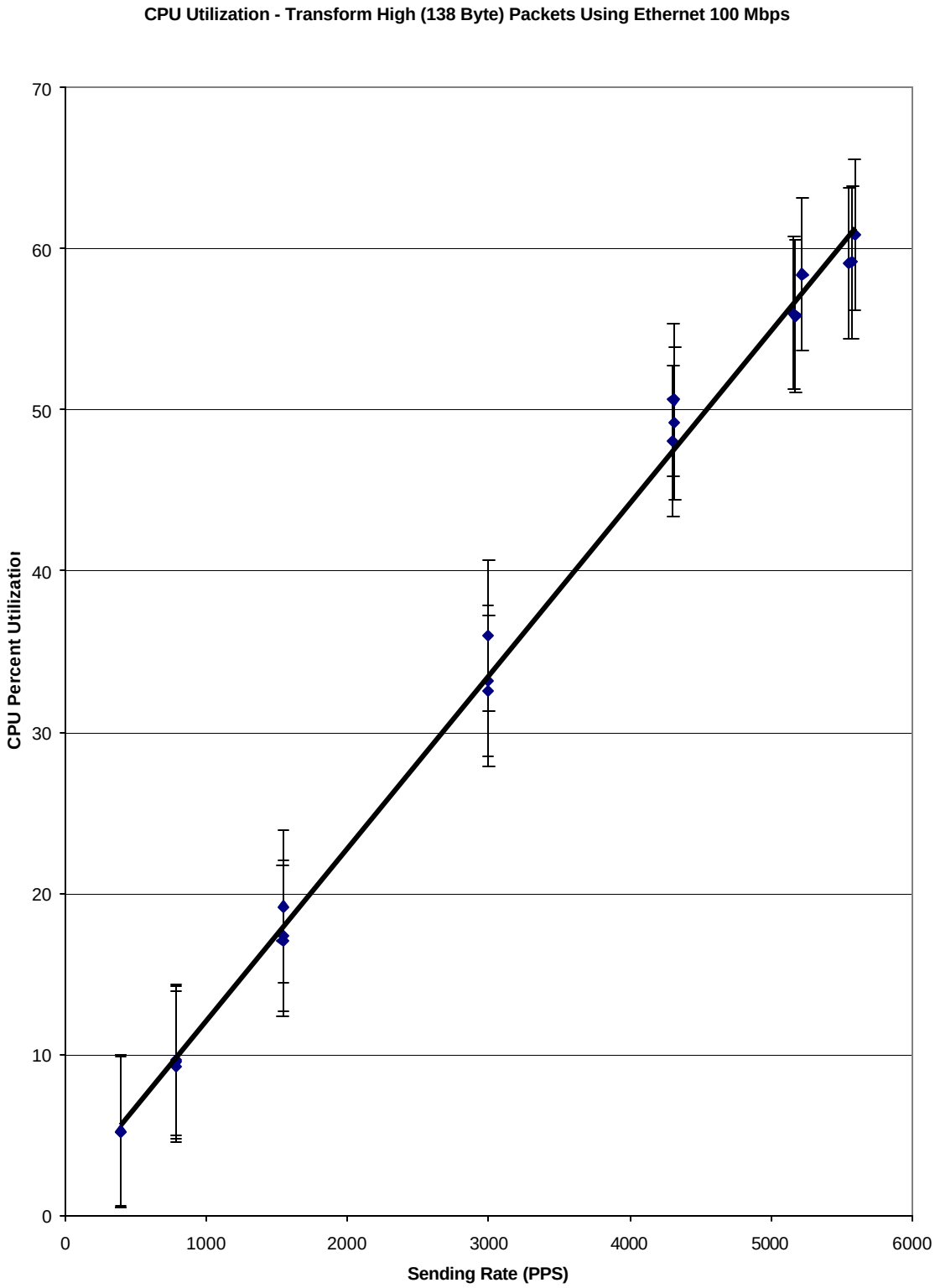

Figure 17. CPU Utilization High-ResolutionTransform Protocol - Ethernet

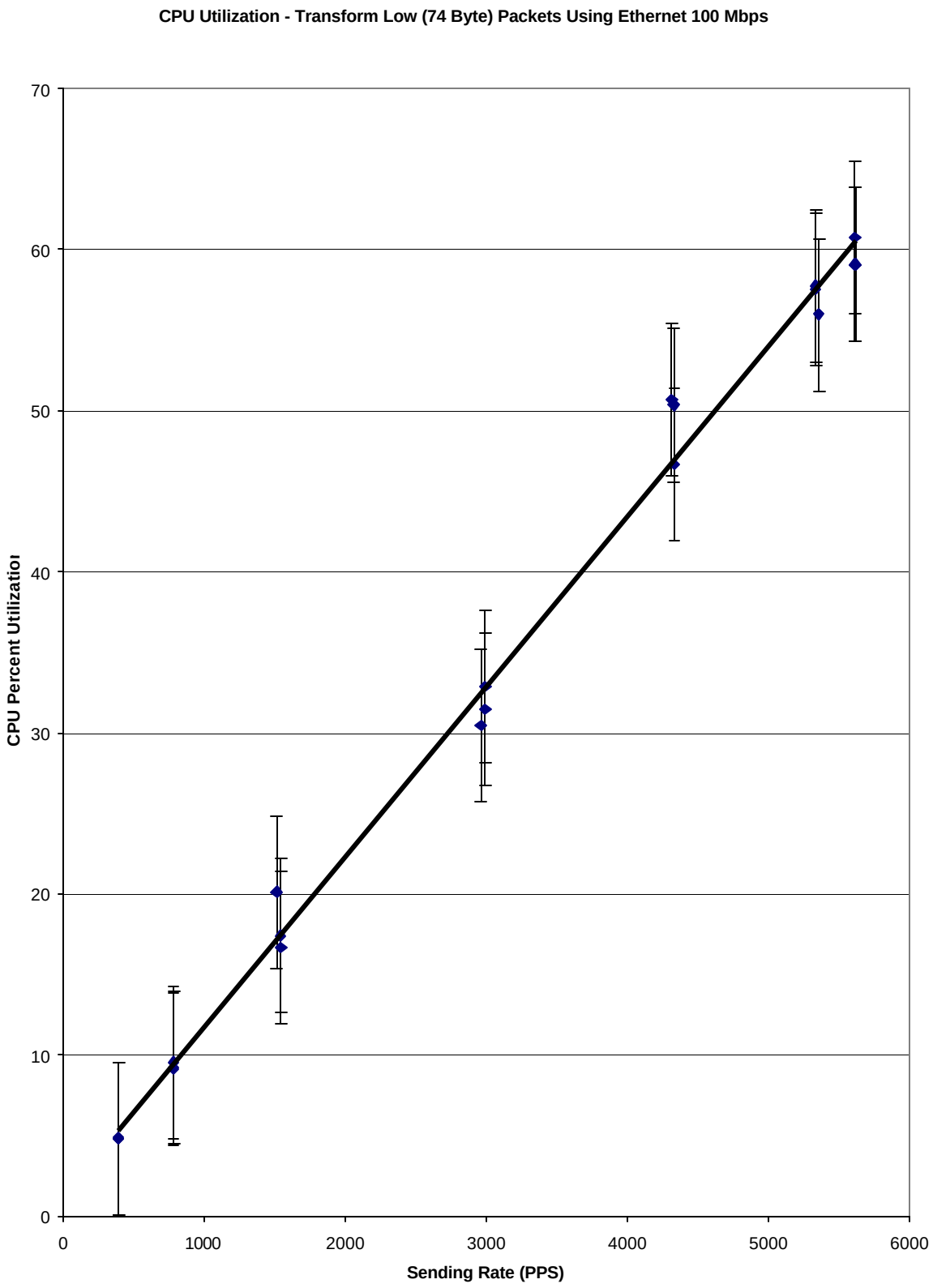

Figure 18. CPU Utilization Low-ResolutionTransform Protocol - Ethernet

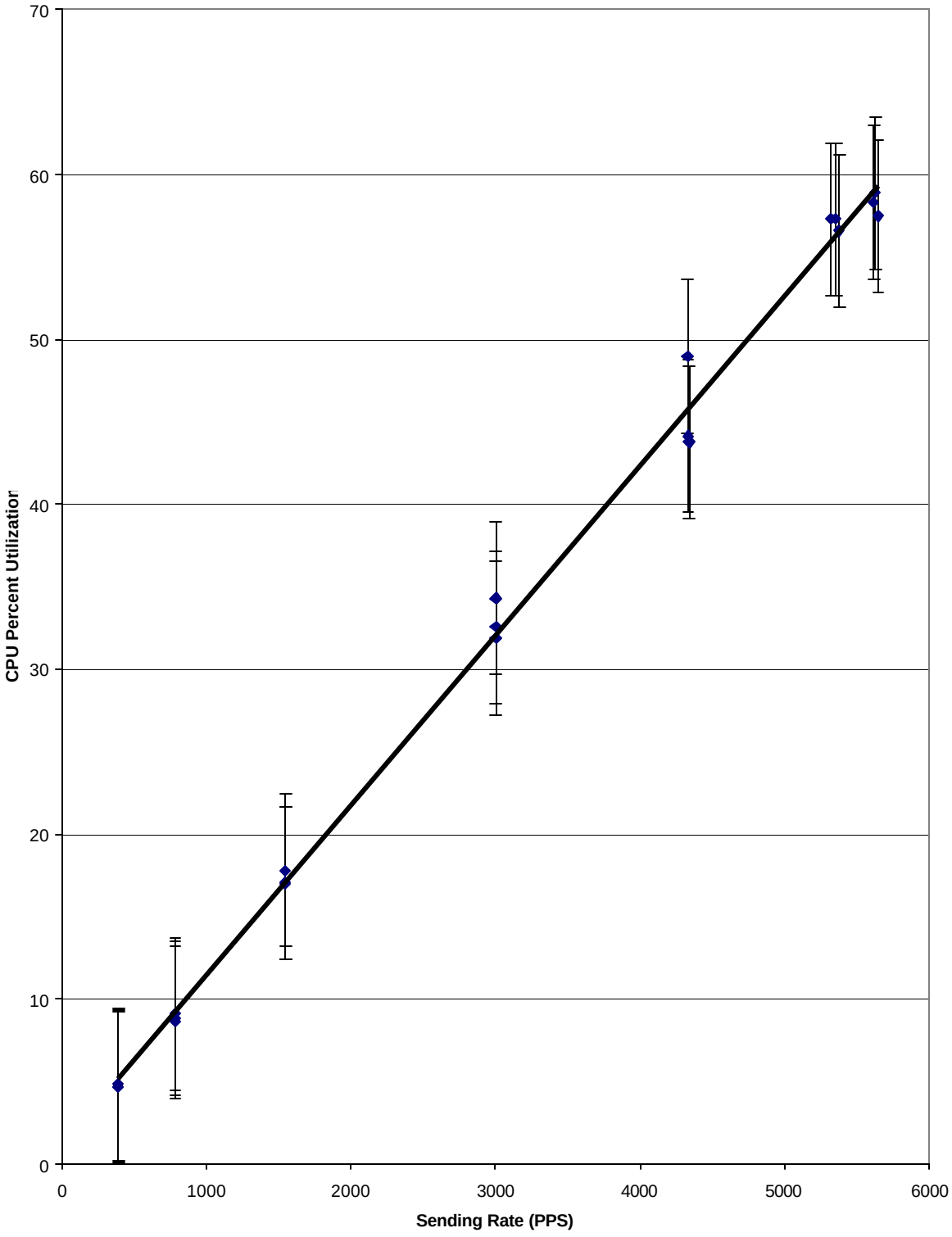

Figure 19. CPU Utilization High-Resolution Animation Protocol - Ethernet

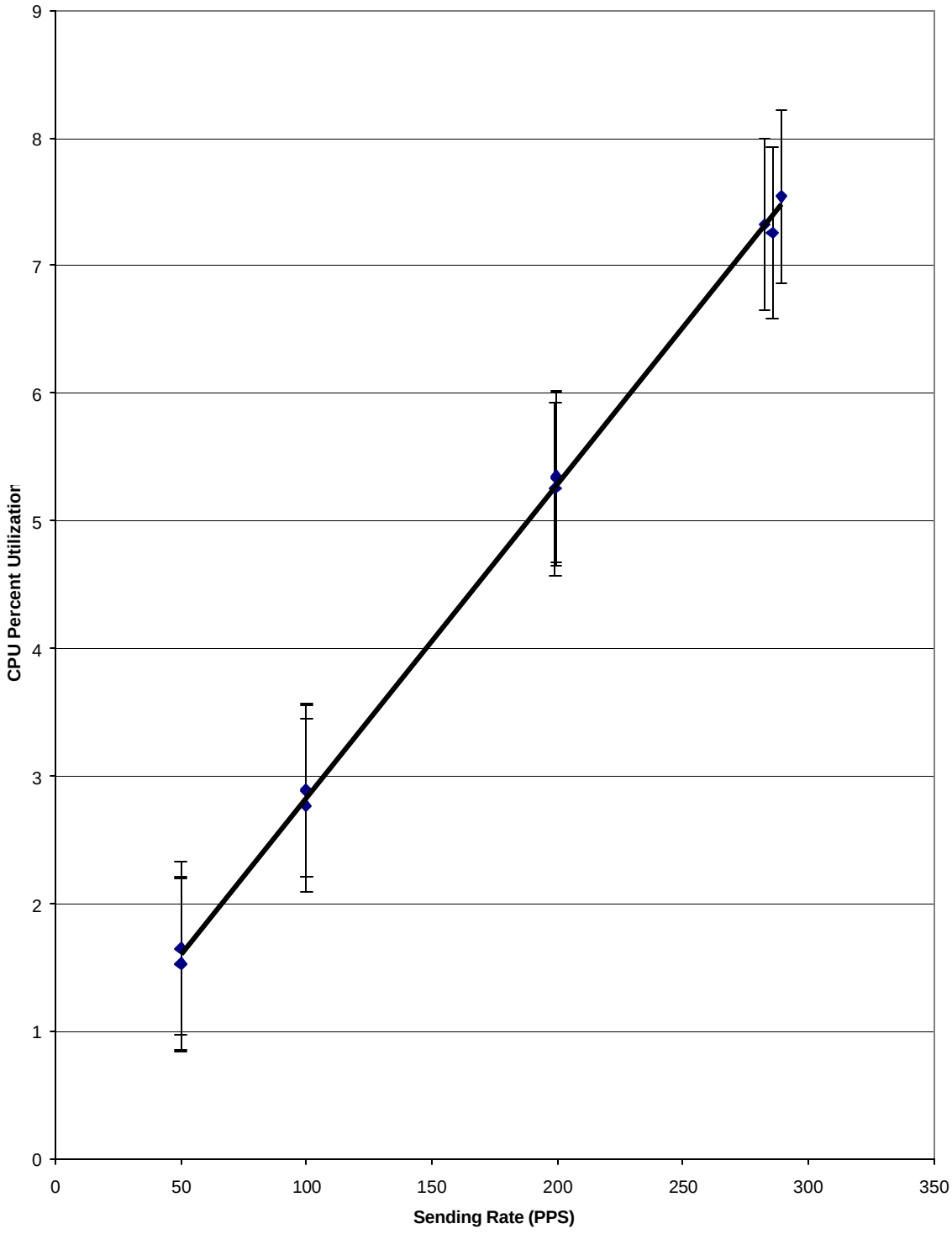

Figure 20. CPU Utilization High-Resolution Transform Protocol - Modem (maximum value 7.5%)

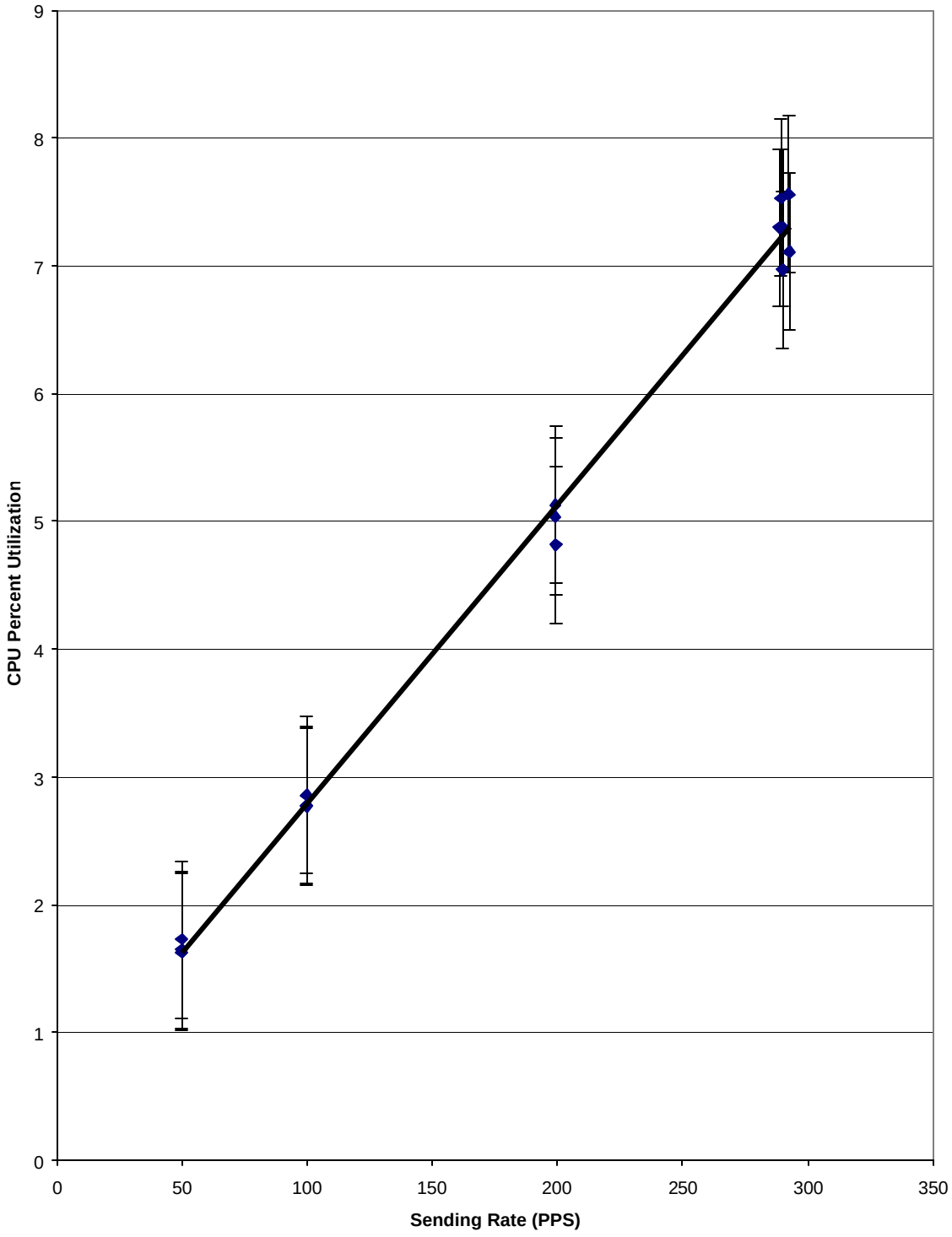

Figure 21. CPU Utilization Low-Resolution Transform Protocol - Modem (maximum value 7.6%)

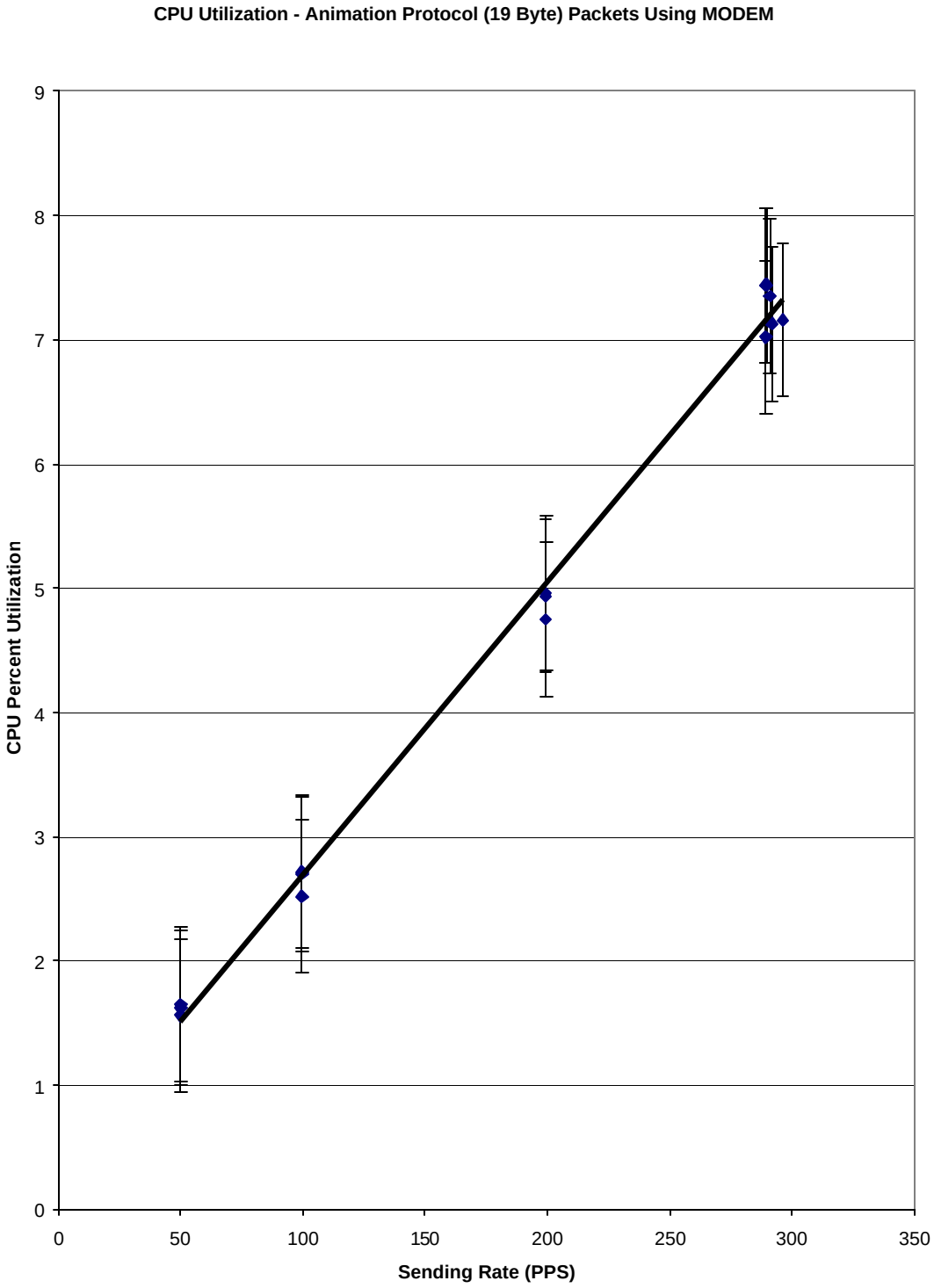

Figure 22. CPU Utilization High-Resolution Animation Protocol - Modem (maximum value 7.5%)

4. Modem CPU Usage

The MODEM CPU usage also has a linear correlation with the sending rate, plotted in Figures 20-22, in packets-per-second. The MODEM CPU usage, like the Ethernet CPU usage, is mostly independent of the tested packet sizes. The CPU usage reaches a maximum of around 3.5% at the maximum packet sending rate of 94 packets-per-second (animation high protocol).

5. Ethernet Cyclic Behavior

CPU utilization is observed to have a spike approximately every 3 ½ minutes. This occurs only while the receiving machine was actually receiving packets sent by NPSNET-V. The apparent cause for this occurrence is Java garbage collection. In the case of low data rates being sent, the spike is pronounced, and with high data rates the CPU utilization takes on a sinusoidal appearance.

This cyclic behavior caused many measurement inconsistencies in earlier tests. Several unsuccessful attempts were made to either control or predict the garbage collection. Since this phenomenon occurs during actual implementation in a virtual environment as well as during experimental testing, all of the test runs were redone and the data-collection interval was increased to four minutes in order to ensure consistent inclusion of a solitary garbage collection event. This strategy ensures more realistic conclusions, since the garbage collection occurs consistently during simulation tests.

6. Modem Packets Per Second

The number of packets per second sent by modem were expected to be around 30 maximum, not the observed 300. The high-resolution transform protocol was used to test this surprising initial result further. The high-resolution packet is 138 bytes with 10

additional bytes prepended by NPSNET-V. UDP adds 8 bytes, IP 20 bytes and PPP 18 bytes as header information. This results in a total of 198 bytes for each packet. If the entire 56K bps is utilized, 7000 bytes can theoretically be sent across the wire per second. This means an estimated 35 200-byte packets per second might be sent for this test. Thus the 300-packet result is quite surprising.

Modem compression hardware is designed to maximize bandwidth capacity limitations. The headers are compressed, as is the payload. The initial payload consisted of byte arrays filled with the default values of zero. Next several byte arrays were randomly created with random values and the sending application would switch between them. Again, surprisingly, this yielded the same results. Finally, in another approach, each byte array was uniquely created with random generation, which finally resulted in an expected maximum sending rate of 30 packets-per-second. Thus the expected modem capacity matched theoretical predictions, and the demonstrated capabilities of modem compression on repetitive traffic is quite impressive.

E. SUMMARY

This chapter discusses the experimental results and analyses used to conduct all three major portions of this thesis. The DBP protocols are written in Java, and each have supporting XML files that describe the data type definitions. The network monitor reports the current state of the network to indicate when to switch resolutions with a DBP protocol. The network testing is designed to establish the optimization points in the NPSNET-V networking architecture, and identifies where to set the network monitor state thresholds.

VI. CONCLUSIONS

A. INTRODUCTION

This chapter presents the conclusions from the experiments performed and analyses derived in this thesis. Two interesting opportunities for future work are presented that can further enhance the optimization of virtual environment networking.

B. ANALYSIS

1. CPU Utilization

There is a direct correlation between CPU Utilization and the sending rate in packets-per-second. For the (1 Ghz) machines used to conduct the experiments using NPSNET-V, the relationship on an Ethernet connection can be estimated by $y = 0.0114x$, where y is the CPU percent utilization (0-100%) and x is the number of packets per second sent to the receiving machine.

Similarly the direct correlation between CPU Utilization and packets per second sent for modem NPSNET-V connections can be estimated. The correlation can be estimated with: $y = 0.025x$, where y is the CPU Utilization percentage (0-100%) and x is the number of packets per second sent.

The initial objective for CPU utilization was to limit the percent used for networking to 10%. For Ethernet connections, this equates to a limit of 877 packets of interest per second. For modem connections, this is estimated at 400 packets per second, which exceeds the maximum achieved rate of 94 packets per second. The network busy threshold is set at 850 packets-per-second for ethernet as a default value, based on these findings. For modem connections, this is estimated at 94 packets per second.

The low threshold is set initially at 850 packets-per-second. The Network Monitor recalculates this value during execution to reflect the average value, in packets-per-second, along with the standard deviation of the mean. Setting the threshold this way will continue to report the network state as busy until the traffic has resumed a normal flow rate. This will help prevent thrashing during surges. Further feedback may be necessary to enable proper participation by modem-connected hosts.

2. Packet Loss

The maximum packet loss observed is around 4%, which is surprisingly low. Keeping with the original requirement of limiting the CPU utilization for networking to 10%, the number of packets per second is 877 maximum for Ethernet connections. This results in an approximately 0.1% packet loss. For modem connections, there has been no observed packet loss. These findings indicate that significantly large distributed virtual environments may be possible for modem-connected participants.

3. Packet Size

The packet size had minimal (if any) impact on the tested metrics for Ethernet connections. For modem connections, the packet size affects the sending rate by causing the modem send buffer to fill quicker for larger packets. The CPU utilization and packet loss are mainly affected by the rate of packet sending, especially for Ethernet connections. For this reason, packet aggregation ought to be considered to limit the total number of packets sent across the network. Reducing packet rates further reduces the possibility of router congestion.

4. DBP Protocols

The DBP protocols are easy to use and there were no problems switching between XML documents for any of the protocols. Switching between high-resolution and low-resolution protocols works effectively. Nevertheless, for these experiments changing the resolution has little impact on network optimization. Combined with packet aggregation or under congestion conditions, being able to specify packet sizes with XML described documents might be a great contribution towards automatic network optimization. Further testing on more heavily utilized networks will likely yield further insights.

The XML-described protocols are easy to edit at runtime using any text editor. Similarly, such protocols can be authored or modified by network-aware software applications, including software agents. The data types declared in the XML document must be supported by the accompanying Java class file. Data types that support the existing protocols are now implemented and work successfully during execution. Although the current experimental implementation does not gain much by reducing packet sizes, tremendous design and implementation advantages can be gained by the runtime extensibility offered by XML-described protocols.

C. FUTURE WORK

There are two principle areas of future work: network management and multi-agent monitoring. Both of these areas outline improvements to application-based network optimization.

1. Network Manager

Instead of implementing a network management scheme that simply reacts to basic high-resolution and low-resolution thresholds, a more sophisticated application-oriented network manager might be implemented. Instead of making adjustments based

on single-period observations, such as an increase in average packets-per-second, automated decisions can be made based on trends observed as a result of key parameter combinations, for example an increase in packets-per-second combined with out-of-order delivery may indicate oncoming congestion that demands immediate attention (i.e. better inferences can be made than merely using a simple observation of an increase in packets-per-second).

The networking parameters currently monitored are packets-per-second and bytes-per-second. Unfortunately, direct measurements of overall CPU utilization were not possible using native Java libraries. Nevertheless, based on manual measurements and correlations established by experimentation, estimates of CPU utilization and packet loss can be achieved. There are also many other network parameters that can be measured. These include buffer sizes, out-of-order packet arrivals, latency, jitter (i.e. latency variations), and particular performance considerations based on the specific application being used.

Like the current implementation, an advanced network manager can provide constantly updated information to the Area of Interest Manager (AOIM) about the network status. The AOIM can then make better-informed optimization decisions. If the best solution includes networking changes, the network manager is notified and directed to implement it. Such a methodology can even occur in a distributed fashion across all participants in a distributed virtual environment.

The proposed design motivation for this network manager is the "better, faster, cheaper: pick any two" paradigm. Better solutions can include greater resolutions, fidelity and correctness. Faster solutions include increased speeds for computation and

network transmissions. Cheaper solutions include lightweight solutions such as smaller payloads, less computationally intense payload algorithms, or further sacrifices in either of the other two solution strategies. Automating analysis and optimizations of such tradeoffs is a rich area for future work.

2. Multi-Agent Monitor

NPSNET-V provides an architectural framework on which to run a virtual environment. The current networking solutions may not work with a future virtual environment. A networking monitor created as part of a Multi-Agent System might learn how to optimize such a distributed system and implement an optimized solution.

Agents have the ability to make adjustments to solution strategies based on feedback from the distributed systems they are monitoring. An agent solution is also able to weed through a high number of input parameters and then determine an appropriate networking strategy.

The proposed approach with which to implement a multi-agent monitor is to populate each system with an organization of agents that gather their information through the Entity Dispatcher. These agents might be organized as advisors to a central agent. The central agent can choose courses of action based on the current priorities and also recommendations by its advisor agents. Such a central agent (or set of agents) will likely have the authority to direct or advise AOIMs as well.

An experimentally grounded and dynamically scalable approach, similar to the monitoring architecture presented in this thesis, will enable the creation and maintenance of rich and responsive large-scale virtual environments.

D. CONTACT INFORMATION

To obtain an electronic copy of the NPSNET-V code, contact one of the Research Faculty through the website at http://www.npsnet.org/~npsnet/v/

APPENDIX A

A. INTRODUCTION

This appendix provides the Java and XML source code for the DBP Transform protocol.

B. DBP TRANSFORM JAVA SOURCE CODE

```
package org.npsnet.v.test.dabp.dbpfoundation.dbputil;

import java.io.*;
import java.net.*;
import java.util.*;
import java.util.Vector.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import java.lang.*;

import org.npsnet.v.sys.*;
import org.npsnet.v.net.*;
import org.npsnet.v.test.dabp.dbpfoundation.*;
import org.web3d.vrtp.dabp.*;
import org.web3d.vrtp.datatypes.*;

/**
 * A protocol for passing information about <code>Transformable</code>
entities.
 * * @author Bill Fischer
 */

public class DBPTransformProtocol extends org.npsnet.v.sys.Protocol
implements                 Serializable
{
    /**
     * The class's version string.
     */
    private static final String classVersion = "$Revision: 1.2 $";

    /**
     * The known location of this class.
     */
    private static final String classCodebaseString =
"http://homer.cs.nps.navy.mil/remote/";

    /**
     * The singleton instance of this class.
     */
    private static DBPTransformProtocol transformProtocol = null;

    /**
     * The known location of this class.
     */
    private static final String PROTO_URL =
"http://homer.cs.nps.navy.mil/remote/";

    /**
     * The URL to the XML file describing the High Resolution packet
layout
```

```
             */
            String hiXmlFile = new
        String("c:/npsnetV/org/npsnet/v/test/dabp/TransformProtocolPacket.xml");

            /**
             * The URL to the XML file describing the High Resolution packet
layout
             */
            String loXmlFile = new
        String("c:/npsnetV/org/npsnet/v/test/dabp/TransformProtocolPacketLoRes.xm
l");

            /**
             * DABP protocols for high resolution
             */
            private org.web3d.vrtp.dabp.Protocol hiResDabpProtocol = new
org.web3d.vrtp.dabp.Protocol(hiXmlFile);

            /**
             * DABP protocols for high resolution
             */
            private org.web3d.vrtp.dabp.Protocol loResDabpProtocol = new
org.web3d.vrtp.dabp.Protocol(loXmlFile);

            /**
             * Returns this class's version string.
             */
            public String getClassVersion() {
                return classVersion;
            }

            /**
             * Returns this class's codebase.
             */
            public URL getClassCodebase() {
                try { return new URL(classCodebaseString); }
                catch(MalformedURLException e) { return null; }
            }

            /**
             * Returns the singleton instance of <code>TransformProtocol</code>.
             */
            public static DBPTransformProtocol getInstance() {
                if(transformProtocol!=null) return transformProtocol;
                else {
                    try {
                        org.npsnet.v.sys.Protocol tmpProt =

getSingleton("org.npsnet.v.test.dabp.dbpfoundation.dbputil.DBPTransformProtocol
",classVersion);
                        if(tmpProt == null)
                            transformProtocol = new DBPTransformProtocol();
                        else
                            transformProtocol = (DBPTransformProtocol)tmpProt;
                    }
                    catch(IDServerException ise) {
                        System.out.println(ise);
                        System.exit(1);
                    }

                    return transformProtocol;
                }
            }
```

```
            /**
             * Constructor; takes a few parameters, which allow us
             * to operate in the world.<p>
             *
             * Note that this constructor is for use with new EntityMasters--
             * We create a new entity and send out information on it. The
             * creation of the entity registers a new entity ID and new
protocolIDs
             * to go along with it.<p>
             *
             * @param pEntity entity that we update
             */
            public DBPTransformProtocol() throws IDServerException
            {
                // A new instance requires a new ID
                   super("DBPTransformProtocol", true);
            }

            /**
             * Constructor. Note that this constructor is for use with
             * things we have discovered from the net. we get in a new
             * message with an existing ID. That means we don't have to
             * look up the ID on the IDServer.<p>
             *
             * @param pID id of the protocol, gleaned from network traffic
             */
            public DBPTransformProtocol(long pID) throws IDServerException
            {
                super(pID, IDCache.getIDCache().getNameForID(pID));
            }

            /**
             * This receives a packet from the entity and serializes it, ie,
changes
             * all the data fields into a byte array that we can send across the
             * network. ADU is some nominal superclass from DABP that subsumes
all
             * packets.
             */
            public byte[] serializeADU(Object pPacketData)
            {
              byte                  serializedData[] = null;
              ADUData               transformADU = null;
              transformADU = (ADUData)pPacketData;
              ByteArrayOutputStream oStream = new ByteArrayOutputStream();
               DataOutputStream dos = new DataOutputStream(oStream);
               transformADU.serialize(dos);
                serializedData = oStream.toByteArray();
                return serializedData;
            }

            /**
             * Receives a packet from the EntityDispatcher (or whatever.) The
             * data we get in is just an array of bytes--we have to interpret
those
             * bytes so that we can do something with them. This consists of two
             * things: determing syntax, so we can retrieve the field named
             * "position.x"; and determining semantics, so we know that
position.x
             * refers to the location of the entity in 3-space. We instantiate
             * an ADUData packet, lay it out in the format described by the
             * protocol's XML document, and fill in the packet's fields from the
```

```
             * incoming byte stream data.
             */
            public void receivePacket(byte[] pData, NetworkAware entity)
            {
            ByteArrayInputStream bais        = new ByteArrayInputStream(pData);

            // Temporary workaround; prevents EntityMasters from being
            // modified remotely
            if(entity instanceof EntityMaster) return;
            if(entity instanceof DBPTransformableMaster) return;

            // We only get packets of one type, so we don't need to check
            // on format beforehand (famous last words).
             ADUData packet = null;
             packet = hiResDabpProtocol.binaryDataToADU(pData); //binaryDataToADU
            try
            {
            if( ((UnsignedByte)(packet.get("markerValue"))).byteValue() == 98)
              {

if(((UnsignedByte)(packet.get("packetResolution.resolution"))).byteValue() ==
1)
             {  packet = loResDabpProtocol.binaryDataToADU(pData);
//binaryDataToADU
             }// end if
              // some temporary variables
              double transformArray [] = new double[16];
              double value1, value2, value3, value4, value5, value6, value7,
value8,
                     value9, value10, value11, value12, value13, value14,
value15,
                     value16;
              int count = 0;
              String intString;
              String fields = "ArrayValues.value";
              Object arrayVals;

              while(count <= 15)
              {
                intString = Integer.toString(count);
                fields = fields + intString;
                arrayVals = packet.get(fields);
                transformArray[count] =
((PrimitiveNumber)arrayVals).doubleValue();
                fields = "ArrayValues.value";
                count++;
              }//end while

              Transform3D transform = new Transform3D(transformArray);
              ((DBPTransformable)entity).setTransform(transform);

       //      System.out.println("Receive side --
TransformProtocol:receivePacket():packet ==     "+ packet);
             }// end if
               else
            { System.err.println("Wrong marker type (not 98)");
            }// end else

            }// end try
            catch(FieldNotFoundException fnfe)
              { System.err.println("Field not found! " + fnfe);
              }// end catch
            return;
```

```
          }// end receivePacket()

          /**
           * Invoked when an entity is modified.  The entity's transform
information
           * is loaded into an ADUData packet, which is then serialized and
passed
           * to the Entity Master.
           */
          public void entityChanged(Entity e, Object info)
          {
          if(e instanceof DBPTransformableGhost) return;

            ADUData transformADU = null;
            transformADU = new ADUData(new
ADU(hiResDabpProtocol.getSchemaNamed("TPPPDU")));

            if(resolution == 1)
            { transformADU = new ADUData(new
ADU(loResDabpProtocol.getSchemaNamed("TPPPDU")));
            }

            //load Vector3d position
            double[] transformArray = new double[16];

            Transform3D transform = new Transform3D();
            ((DBPTransformable)e).getTransform(transform);
            transform.get(transformArray);

            int count = 0;
            String intString;
            String fields = "ArrayValues.value";

              //write the resolution value
            try
            {
              transformADU.put("packetResolution.resolution", new
UnsignedByte(resolution));
            }// end try
            catch(FieldTypeException fte)
            {  System.err.println("Field Type not found: " + fte);
            }// end catch
            catch(FieldNotFoundException fnfe)
            {  System.err.println("Field not found: " + fnfe);
            }// end catch
              while(count <= 15)
              {
                intString = Integer.toString(count);
                fields = fields + intString;
            try
            {
                // giving up some efficiency here to maximize flexibility &
robustness.
                // Can set the fields types individually by type in the XML
document
                if(transformADU.get(fields) instanceof DoublePrecision)
                  transformADU.put(fields, new
DoublePrecision(transformArray[count]));
                else if(transformADU.get(fields) instanceof SinglePrecision)
                  transformADU.put(fields, new
SinglePrecision((float)transformArray[count]));

            }// end try
```

```
            catch(FieldTypeException fte)
            {  System.err.println("Field Type not found: " + fte);
            }// end catch
            catch(FieldNotFoundException fnfe)
            {  System.err.println("Field not found: " + fnfe);
            }// end catch
                fields = "ArrayValues.value";
                count++;
              }//end while

       //      System.out.println("Send side ---
TransformProtocol:entityChanged():transformADU == " +
transformADU);//*************888888888888888888

           // first time thru and field values may be null
              if(transformADU.getAduName() == "") return;

            // Serialize and send transform data for the entity
             byte[] data = serializeADU(transformADU);

            Set distroSet = AOIM.getAOIM().getDistributionSet(e, this);

            EntityDispatcher.getEntityDispatcher().sendPacket(this,
                                                              e.getID(),
                                                              distroSet,
                                                              data);
           }// end entityChanged()

       };
```

C. DBP TRANSFORM HIGH RESOLUTION XML SOURCE DOCUMENT

```
       <?xml version="1.0"?>

       <!DOCTYPE Protocol SYSTEM "./DynamicBehaviorProtocol.dtd">

       <Protocol name="TRANSFORM_PACKET"
                 markerPosition="0"
                 markerType="org.web3d.vrtp.datatypes.UnsignedByte"
                 clientStub="http://some.url.com/baz">

       <Type name="org.web3d.vrtp.datatypes.DoublePrecision"
value="file://org.web3d.vrtp.datatypes.DoublePrecision"></Type>
       <Type name="org.web3d.vrtp.datatypes.UnsignedByte"
value="file://org.web3d.vrtp.datatypes.UnsignedByte"></Type>
       <Type name="org.web3d.vrtp.datatypes.SignedInteger"
value="file://org.web3d.vrtp.datatypes.SignedInteger"></Type>

       <Channel multicastAddress="225.6.9.121"
                       multicastPort = "1616" />

       <PacketHeader name="TPP">

         <Field name="sequence"
                type="org.web3d.vrtp.datatypes.SignedInteger"
                initialValue="0"/>

         <Field name="timestamp"
                type="org.web3d.vrtp.datatypes.SignedInteger"
                initialValue="0"/>
```

```
</PacketHeader>

<ADU name="TPPPDU" markerValue="98">

  <Field name="markerValue"
         type="org.web3d.vrtp.datatypes.UnsignedByte"
         initialValue="98"/>

<Structure name="packetResolution">

          <Field name="resolution"
          type="org.web3d.vrtp.datatypes.UnsignedByte"
          initialValue="2"/>

</Structure>

<Structure name="ArrayValues">

      <Field name="value0"
       type="org.web3d.vrtp.datatypes.DoublePrecision"
       initialValue="0"/>

      <Field name="value1"
       type="org.web3d.vrtp.datatypes.DoublePrecision"
       initialValue="0"/>

      <Field name="value2"
       type="org.web3d.vrtp.datatypes.DoublePrecision"
       initialValue="0"/>

      <Field name="value3"
       type="org.web3d.vrtp.datatypes.DoublePrecision"
       initialValue="0"/>

      <Field name="value4"
       type="org.web3d.vrtp.datatypes.DoublePrecision"
       initialValue="0"/>

      <Field name="value5"
       type="org.web3d.vrtp.datatypes.DoublePrecision"
       initialValue="0"/>

      <Field name="value6"
       type="org.web3d.vrtp.datatypes.DoublePrecision"
       initialValue="0"/>

      <Field name="value7"
       type="org.web3d.vrtp.datatypes.DoublePrecision"
       initialValue="0"/>

      <Field name="value8"
       type="org.web3d.vrtp.datatypes.DoublePrecision"
       initialValue="0"/>

      <Field name="value9"
       type="org.web3d.vrtp.datatypes.DoublePrecision"
       initialValue="0"/>

      <Field name="value10"
       type="org.web3d.vrtp.datatypes.DoublePrecision"
       initialValue="0"/>

      <Field name="value11"
```

```
        type="org.web3d.vrtp.datatypes.DoublePrecision"
        initialValue="0"/>

      <Field name="value12"
        type="org.web3d.vrtp.datatypes.DoublePrecision"
        initialValue="0"/>

      <Field name="value13"
        type="org.web3d.vrtp.datatypes.DoublePrecision"
        initialValue="0"/>

      <Field name="value14"
        type="org.web3d.vrtp.datatypes.DoublePrecision"
        initialValue="0"/>

      <Field name="value15"
        type="org.web3d.vrtp.datatypes.DoublePrecision"
        initialValue="0"/>

</Structure>

</ADU>

</Protocol>
```

D. DBP TRANSFORM LOW RESOLUTION XML SOURCE DOCUMENT

```
<?xml version="1.0"?>

<!DOCTYPE Protocol SYSTEM "./DynamicBehaviorProtocol.dtd">

<Protocol name="TRANSFORM_PACKET"
          markerPosition="0"
          markerType="org.web3d.vrtp.datatypes.UnsignedByte"
          clientStub="http://some.url.com/baz">

<Type name="org.web3d.vrtp.datatypes.UnsignedByte"
value="file://org.web3d.vrtp.datatypes.UnsignedByte"></Type>
<Type name="org.web3d.vrtp.datatypes.SignedInteger"
value="file://org.web3d.vrtp.datatypes.SignedInteger"></Type>
<Type name="org.web3d.vrtp.datatypes.SinglePrecision"
value="file://org.web3d.vrtp.datatypes.SinglePrecision"></Type>

<Channel multicastAddress="225.6.9.121"
                multicastPort = "1616" />

<PacketHeader name="TPP">

  <Field name="sequence"
         type="org.web3d.vrtp.datatypes.SignedInteger"
         initialValue="0"/>

  <Field name="timestamp"
         type="org.web3d.vrtp.datatypes.SignedInteger"
         initialValue="0"/>

</PacketHeader>

<ADU name="TPPPDU" markerValue="98">

  <Field name="markerValue"
          type="org.web3d.vrtp.datatypes.UnsignedByte"
          initialValue="98"/>
```

```
<Structure name="packetResolution">

  <Field name="resolution"
         type="org.web3d.vrtp.datatypes.UnsignedByte"
         initialValue="1"/>

</Structure>

<Structure name="ArrayValues">

      <Field name="value0"
       type="org.web3d.vrtp.datatypes.SinglePrecision"
       initialValue="0"/>

      <Field name="value1"
       type="org.web3d.vrtp.datatypes.SinglePrecision"
       initialValue="0"/>

      <Field name="value2"
       type="org.web3d.vrtp.datatypes.SinglePrecision"
       initialValue="0"/>

      <Field name="value3"
       type="org.web3d.vrtp.datatypes.SinglePrecision"
       initialValue="0"/>

      <Field name="value4"
       type="org.web3d.vrtp.datatypes.SinglePrecision"
       initialValue="0"/>

      <Field name="value5"
       type="org.web3d.vrtp.datatypes.SinglePrecision"
       initialValue="0"/>

      <Field name="value6"
       type="org.web3d.vrtp.datatypes.SinglePrecision"
       initialValue="0"/>

      <Field name="value7"
       type="org.web3d.vrtp.datatypes.SinglePrecision"
       initialValue="0"/>

      <Field name="value8"
       type="org.web3d.vrtp.datatypes.SinglePrecision"
       initialValue="0"/>

      <Field name="value9"
       type="org.web3d.vrtp.datatypes.SinglePrecision"
       initialValue="0"/>

      <Field name="value10"
       type="org.web3d.vrtp.datatypes.SinglePrecision"
       initialValue="0"/>

      <Field name="value11"
       type="org.web3d.vrtp.datatypes.SinglePrecision"
       initialValue="0"/>

      <Field name="value12"
       type="org.web3d.vrtp.datatypes.SinglePrecision"
       initialValue="0"/>
```

```
        <Field name="value13"
         type="org.web3d.vrtp.datatypes.SinglePrecision"
         initialValue="0"/>

        <Field name="value14"
         type="org.web3d.vrtp.datatypes.SinglePrecision"
         initialValue="0"/>

        <Field name="value15"
         type="org.web3d.vrtp.datatypes.SinglePrecision"
         initialValue="0"/>

    </Structure>

    </ADU>

    </Protocol>
```

E. SUMMARY

The source code and source documents presented in this appendix work together to provide both a high and low resolution capability for the DBP transform protocol.

APPENDIX B

A. INTRODUCTION

This appendix provides the Java and XML source code for the DBP Inertia protocol.

B. DBP INERTIA JAVA SOURCE CODE

```
package org.npsnet.v.test.dabp.dbpfoundation.dbputil;

import java.io.*;
import java.net.*;
import java.util.*;
import java.util.Vector.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import java.lang.*;

import org.npsnet.v.sys.*;
import org.npsnet.v.net.*;
import org.npsnet.v.test.dabp.dbpfoundation.*;
import org.npsnet.v.test.dabp.dbpfoundation.dbputil.*;
import org.web3d.vrtp.dabp.*;
import org.web3d.vrtp.datatypes.*;

/**
 * A protocol that passes information about inertial state (position,
 * linear and angular velocity).
 *
 * @author Bill Fischer
 */

public class DBPInertiaProtocol extends org.npsnet.v.sys.Protocol
implements Serializable
{
    /**
     * The class's version string.
     */
    private static final String classVersion = "$Revision: 1.3 $";

    /**
     * The known location of this class.
     */
    private static final String classCodebaseString =
"http://homer.cs.nps.navy.mil/remote/";

    /**
     * The singleton instance of this class.
     */
    private static DBPInertiaProtocol inertiaProtocol;

    /**
     * The known location of this class.
     */
    private static final String PROTO_URL =
"http://homer.cs.nps.navy.mil/remote/";

    /**
```

```
            * The URL to the XML file describing the High Resolution packet
layout
            */
           String hiXmlFile = new
       String("c:/npsnetV/org/npsnet/v/test/dabp/InertiaProtocolPacket.xml");

           /**
            * The URL to the XML file describing the High Resolution packet
layout
            */

           String loXmlFile = new
       String("c:/npsnetV/org/npsnet/v/test/dabp/InertiaProtocolPacketLoRes.xml"
);

           /**
            * DABP protocols for high resolution
            */
           private org.web3d.vrtp.dabp.Protocol hiResDabpProtocol = new
       org.web3d.vrtp.dabp.Protocol(hiXmlFile);

           /**
            * DABP protocols for low resolution
            */
           private org.web3d.vrtp.dabp.Protocol loResDabpProtocol = new
       org.web3d.vrtp.dabp.Protocol(loXmlFile);

           /**
            * Returns this class's version string.
            */
           public String getClassVersion() {
               return classVersion;
           }

           /**
            * Returns this class's codebase.
            */
           public URL getClassCodebase() {
               try { return new URL(classCodebaseString); }
               catch(MalformedURLException e) { return null; }
           }

           /**
            * Returns the singleton instance of <code>TransformProtocol</code>.
            */
           public static DBPInertiaProtocol getInstance() {
               if(inertiaProtocol!=null) return inertiaProtocol;
               else {
                   try {
                       org.npsnet.v.sys.Protocol tmpProt =

getSingleton("org.npsnet.v.test.dabp.dbpfoundation.dbputil.DBPInertiaProtocol",
classVersion);

                       if(tmpProt == null)
                           inertiaProtocol = new DBPInertiaProtocol();
                       else
                           inertiaProtocol = (DBPInertiaProtocol)tmpProt;
                   }
                   catch(IDServerException ise) {
                       System.out.println(ise);
                       System.exit(1);
                   }
```

```
                return inertiaProtocol;
            }
        }

        /**
         * Constructor; takes a few parameters, which allow us
         * to operate in the world.<p>
         *
         * Note that this constructor is for use with new EntityMasters--
         * We create a new entity and send out information on it. The
         * creation of the entity registers a new entity ID and new
protocolIDs
         * to go along with it.<p>
         *
         * @param pEntity entity that we update
         */
        public DBPInertiaProtocol() throws IDServerException
        {
            // A new instance requires a new ID
               super("DBPInertiaProtocol", true);
        }

        /**
         * constructor. Note that this constructor is for use with
         * things we have discovered from the net. we get in a new
         * message with an existing ID. That means we don't have to
         * look up the ID on the IDServer.<p>
         *
         * @param pID id of the protocol, gleaned from network traffic
         */
        public DBPInertiaProtocol(long pID) throws IDServerException
        {
            super(pID, IDCache.getIDCache().getNameForID(pID));
        }

        /**
         * This receives a packet from the entity and serializes it, ie,
changes
         * all the data fields into a byte array that we can send across the
         * network. ADU is some nominal superclass from DABP that subsumes
all
         * packets.
         */
        public byte[] serializeADU(Object pPacketData)
        {
          byte                  serializedData[] = null;
          ADUData               inertiaADU = null;
          inertiaADU = (ADUData)pPacketData;

          ByteArrayOutputStream oStream = new ByteArrayOutputStream();
          DataOutputStream dos = new DataOutputStream(oStream);
          inertiaADU.serialize(dos);
          serializedData = oStream.toByteArray();
          return serializedData;
        }

        /**
         * Receives a packet from the EntityDispatcher (or whatever.) The
         * data we get in is just an array of bytes--we have to interpret
those
         * bytes so that we can do something with them. This consists of two
         * things: determing syntax, so we can retrieve the field named
```

```
            * "position.x"; and determining semantics, so we know that
position.x
            * refers to the location of the entity in 3-space. We instantiate
            * an ADUData packet, lay it out in the format described by the
            * protocol's XML document, and fill in the packet's fields from the
            * incoming byte stream data.
            */
           public void receivePacket(byte[] pData, NetworkAware entity)
           {
           ByteArrayInputStream bais        = new ByteArrayInputStream(pData);

           // Temporary workaround; prevents EntityMasters from being
           // modified remotely
           if(entity instanceof EntityMaster) return;
           if(entity instanceof DBPInertialMaster) return;

           // We only get packets of one type, so we don't need to check
           // on format beforehand (famous last words).

            ADUData packet = null;
            packet = hiResDabpProtocol.binaryDataToADU(pData); //binaryDataToADU

           // some temporary objects & variables
           Vector3d position = null;
           Vector3d linearVelocity = null;
           Vector3d angularVelocity = null;
           Quat4d orientation = null;
           long posRefTime = 0;
           long orRefTime = 0;
           boolean timeCorrect = true;
           boolean lowResPacket = false;

           try
           {
           if( ((UnsignedByte)(packet.get("markerValue"))).byteValue() == 99)
             {

if(((UnsignedByte)(packet.get("packetResolution.resolution"))).byteValue() ==
1)
             {  packet = loResDabpProtocol.binaryDataToADU(pData);
//binaryDataToADU
             }// end if
             // some temporary variables
             double xPos, yPos, zPos, or1, or2, or3, or4, linX, linY, linZ,
angX, angY, angZ;

             // fill the blank ADUData packet with the corresponding fields from
the
             //    incoming byte stream
             xPos =
(double)(((PrimitiveNumber)packet.get("Position.x")).doubleValue());
             yPos =
(double)(((PrimitiveNumber)packet.get("Position.y")).doubleValue());
             zPos =
(double)(((PrimitiveNumber)packet.get("Position.z")).doubleValue());
             position = new Vector3d(xPos, yPos, zPos);

             PrimitiveNumber posPrim =
       ((PrimitiveNumber)packet.get("posRefTime.posTime"));
             if(posPrim instanceof LongInteger)
               posRefTime = (long)(posPrim.longValue());

             else if(posPrim instanceof UnsignedInteger)
```

```
            {
              long sysTime = System.currentTimeMillis();
              long timeMask = 0xffffffff00000000l;
              sysTime = sysTime & timeMask;
              long tempInt = posPrim.longValue();
              posRefTime = (sysTime | tempInt);
            }// end else if

            PrimitiveNumber orPrim =
((PrimitiveNumber)packet.get("orRefTime.orTime"));
            if(orPrim instanceof LongInteger)
              orRefTime = (long)(posPrim.longValue());
            else if(orPrim instanceof UnsignedInteger)
            {
              long sysTime = System.currentTimeMillis();
              long timeMask = 0xffffffff00000000l;
              sysTime = sysTime & timeMask;
              long tempInt = orPrim.longValue();
              orRefTime = (sysTime | tempInt);
            }// else if

            or1 =
(double)(((PrimitiveNumber)packet.get("orientation.or1")).doubleValue());
            or2 =
(double)(((PrimitiveNumber)packet.get("orientation.or2")).doubleValue());
            or3 =
(double)(((PrimitiveNumber)packet.get("orientation.or3")).doubleValue());
            or4 =
(double)(((PrimitiveNumber)packet.get("orientation.or4")).doubleValue());
            orientation = new Quat4d(or1, or2, or3, or4);

            linX =
(double)(((PrimitiveNumber)packet.get("linearVelocity.x")).doubleValue());
            linY =
(double)(((PrimitiveNumber)packet.get("linearVelocity.y")).doubleValue());
            linZ =
(double)(((PrimitiveNumber)packet.get("linearVelocity.z")).doubleValue());
            linearVelocity = new Vector3d(linX, linY, linZ);

            angX =
      (double)(((PrimitiveNumber)packet.get("angularVelocity.x")).doubleValue()
);
            angY =
      (double)(((PrimitiveNumber)packet.get("angularVelocity.y")).doubleValue()
);
            angZ =
      (double)(((PrimitiveNumber)packet.get("angularVelocity.z")).doubleValue()
);
            angularVelocity = new Vector3d(angX, angY, angZ);

      //      System.out.println("Receive side --
InertiaProtocol:receivePacket():packet == "+    packet);

               ((DBPInertial)entity).setInertialState(position, posRefTime,
orientation,
                                                      orRefTime, linearVelocity,
                                                      angularVelocity, 1);
          }// end if
          else
          { System.err.println("Wrong marker type (not 99)");
          }// end else
          }// end try
          catch(FieldNotFoundException fnfe)
```

```
            {
              System.err.println("Field not found! " + fnfe);
            }// end catch
          return;
          }

          /**
           * Invoked when an entity is modified.  The entity's inertia
information
           * is loaded into an ADUData packet, which is then serialized and
passed
           * to the Entity Master.
           */
          public void entityChanged(Entity e, Object info)
          {
          if(e instanceof DBPInertialGhost) return;

            ADUData inertiaADU = null;

            inertiaADU = new ADUData(new
ADU(hiResDabpProtocol.getSchemaNamed("IPPPDU")));

            if(resolution == 1)
              inertiaADU = new ADUData(new
ADU(loResDabpProtocol.getSchemaNamed("IPPPDU")));

            //load Vector3d position
            double[] positionArray = new double[3];

            Vector3d position = new Vector3d();
            ((DBPInertial)e).getPosition(position);
            position.get(positionArray);

            double xPos = positionArray[0];
            double yPos = positionArray[1];
            double zPos = positionArray[2];

            // orientation
            double[] orientationArray = new double[4];

            Quat4d orientation = new Quat4d();
            ((DBPInertial)e).getOrientation(orientation);
            orientation.get(orientationArray);

            double or1 = orientationArray[0];
            double or2 = orientationArray[1];
            double or3 = orientationArray[2];
            double or4 = orientationArray[3];

            // linear velocity
            double[] linearVelocityArray = new double[3];

            Vector3d linearVelocity = new Vector3d();
            ((DBPInertial)e).getLinearVelocity(linearVelocity);
            linearVelocity.get(linearVelocityArray);

            double linX = linearVelocityArray[0];
            double linY = linearVelocityArray[1];
            double linZ = linearVelocityArray[2];

            // angular velocity
            double[] angularVelocityArray = new double[3];
```

```
            Vector3d angularVelocity = new Vector3d();
            ((DBPInertial)e).getAngularVelocity(angularVelocity);
            angularVelocity.get(angularVelocityArray);

            double angX = angularVelocityArray[0];
            double angY = angularVelocityArray[1];
            double angZ = angularVelocityArray[2];
            try
            {
            // load the ADUData packet
              inertiaADU.put("packetResolution.resolution", new
UnsignedByte(resolution));

              String tempString = new String("Position.x");
              if(inertiaADU.get(tempString) instanceof DoublePrecision)
                inertiaADU.put(tempString, new DoublePrecision(xPos));
              else if(inertiaADU.get(tempString) instanceof SinglePrecision)
                inertiaADU.put(tempString, new SinglePrecision((float)xPos));

              tempString = new String("Position.y");
              if(inertiaADU.get(tempString) instanceof DoublePrecision)
                inertiaADU.put(tempString, new DoublePrecision(yPos));
              else if(inertiaADU.get(tempString) instanceof SinglePrecision)
                inertiaADU.put(tempString, new SinglePrecision((float)yPos));

              tempString = new String("Position.z");
              if(inertiaADU.get(tempString) instanceof DoublePrecision)
                inertiaADU.put(tempString, new DoublePrecision(zPos));
              else if(inertiaADU.get(tempString) instanceof SinglePrecision)
                inertiaADU.put(tempString, new SinglePrecision((float)zPos));

              tempString = new String("posRefTime.posTime");
              if(inertiaADU.get(tempString) instanceof LongInteger)
                inertiaADU.put(tempString, new
LongInteger(System.currentTimeMillis()));

              else if(inertiaADU.get(tempString) instanceof UnsignedInteger)
              {
                long sysTime = System.currentTimeMillis();
                sysTime = sysTime & 0x00000000ffffffffl;
                int tempInt = (int)sysTime;
                UnsignedInteger usIntTime = new UnsignedInteger(tempInt);
                inertiaADU.put(tempString, usIntTime);
              }// end else if

              tempString = new String("orRefTime.orTime");
              if(inertiaADU.get(tempString) instanceof LongInteger)
                inertiaADU.put(tempString, new
LongInteger(System.currentTimeMillis()));
              else if(inertiaADU.get(tempString) instanceof UnsignedInteger)
              {
                long sysTime = System.currentTimeMillis();
                sysTime = sysTime & 0x00000000ffffffffl;
                int tempInt = (int)sysTime;
                UnsignedInteger usIntTime = new UnsignedInteger(tempInt);
                inertiaADU.put(tempString, usIntTime);
              }// end else if

              tempString = new String("orientation.or1");
              if(inertiaADU.get(tempString) instanceof DoublePrecision)
                inertiaADU.put(tempString, new DoublePrecision(or1));
              else if(inertiaADU.get(tempString) instanceof SinglePrecision)
                inertiaADU.put(tempString, new SinglePrecision((float)or1));
```

```
  tempString = new String("orientation.or2");
  if(inertiaADU.get(tempString) instanceof DoublePrecision)
    inertiaADU.put(tempString, new DoublePrecision(or2));
  else if(inertiaADU.get(tempString) instanceof SinglePrecision)
    inertiaADU.put(tempString, new SinglePrecision((float)or2));

  tempString = new String("orientation.or3");
  if(inertiaADU.get(tempString) instanceof DoublePrecision)
    inertiaADU.put(tempString, new DoublePrecision(or3));
  else if(inertiaADU.get(tempString) instanceof SinglePrecision)
    inertiaADU.put(tempString, new SinglePrecision((float)or3));

  tempString = new String("orientation.or4");
  if(inertiaADU.get(tempString) instanceof DoublePrecision)
    inertiaADU.put(tempString, new DoublePrecision(or4));
  else if(inertiaADU.get(tempString) instanceof SinglePrecision)
    inertiaADU.put(tempString, new SinglePrecision((float)or4));

  tempString = new String("linearVelocity.x");
  if(inertiaADU.get(tempString) instanceof DoublePrecision)
    inertiaADU.put(tempString, new DoublePrecision(linX));
  else if(inertiaADU.get(tempString) instanceof SinglePrecision)
    inertiaADU.put(tempString, new SinglePrecision((float)linX));

  tempString = new String("linearVelocity.y");
  if(inertiaADU.get(tempString) instanceof DoublePrecision)
    inertiaADU.put(tempString, new DoublePrecision(linY));
  else if(inertiaADU.get(tempString) instanceof SinglePrecision)
    inertiaADU.put(tempString, new SinglePrecision((float)linY));

  tempString = new String("linearVelocity.z");
  if(inertiaADU.get(tempString) instanceof DoublePrecision)
    inertiaADU.put(tempString, new DoublePrecision(linZ));
  else if(inertiaADU.get(tempString) instanceof SinglePrecision)
    inertiaADU.put(tempString, new SinglePrecision((float)linZ));

  tempString = new String("angularVelocity.x");
  if(inertiaADU.get(tempString) instanceof DoublePrecision)
    inertiaADU.put(tempString, new DoublePrecision(angX));
  else if(inertiaADU.get(tempString) instanceof SinglePrecision)
    inertiaADU.put(tempString, new SinglePrecision((float)angX));

  tempString = new String("angularVelocity.y");
  if(inertiaADU.get(tempString) instanceof DoublePrecision)
    inertiaADU.put(tempString, new DoublePrecision(angY));
  else if(inertiaADU.get(tempString) instanceof SinglePrecision)
    inertiaADU.put(tempString, new SinglePrecision((float)angY));

  tempString = new String("angularVelocity.z");
  if(inertiaADU.get(tempString) instanceof DoublePrecision)
    inertiaADU.put(tempString, new DoublePrecision(angZ));
  else if(inertiaADU.get(tempString) instanceof SinglePrecision)
    inertiaADU.put(tempString, new SinglePrecision((float)angZ));

}// end try
catch(FieldTypeException fte)
{
  System.err.println("Field Type not found: " + fte);
}// end catch
catch(FieldNotFoundException fnfe)
{
  System.err.println("Field not found: " + fnfe);
```

```
            }// end catch

      //      System.out.println("Send side ---
InertiaProtocol:entityChanged():inertiaADU == \n" + inertiaADU);

            // Serialize and send inertial data for the entity
            // Serialize and send transform data for the entity
            byte[] data = serializeADU(inertiaADU);

            Set distroSet = AOIM.getAOIM().getDistributionSet(e, this);

            EntityDispatcher.getEntityDispatcher().sendPacket(this,
                                                              e.getID(),
                                                              distroSet,
                                                              data);
         }
      };
```

C. DBP INERTIA HIGH RESOLUTION SOURCE DOCUMENT

```
      <?xml version="1.0"?>

      <!DOCTYPE Protocol SYSTEM "./DynamicBehaviorProtocol.dtd">

      <Protocol name="INERTIAL_PACKET"
                markerPosition="0"
                markerType="org.web3d.vrtp.datatypes.UnsignedByte"
                clientStub="http://some.url.com/baz">


      <Type name="org.web3d.vrtp.datatypes.DoublePrecision"
      value="file://org.web3d.vrtp.datatypes.DoublePrecision"></Type>
      <Type name="org.web3d.vrtp.datatypes.LongInteger"
      value="file://org.web3d.vrtp.datatypes.LongInteger"></Type>
      <Type name="org.web3d.vrtp.datatypes.UnsignedByte"
      value="file://org.web3d.vrtp.datatypes.UnsignedByte"></Type>
      <Type name="org.web3d.vrtp.datatypes.SignedInteger"
      value="file://org.web3d.vrtp.datatypes.SignedInteger"></Type>

      <Channel multicastAddress="225.4.3.122"
                      multicastPort = "1717" />

      <PacketHeader name="RTP">

        <Field name="sequence"
               type="org.web3d.vrtp.datatypes.SignedInteger"
               initialValue="0"/>

        <Field name="timestamp"
               type="org.web3d.vrtp.datatypes.SignedInteger"
               initialValue="0"/>

      </PacketHeader>

      <ADU name="IPPPDU" markerValue="99">

        <Field name="markerValue"
                type="org.web3d.vrtp.datatypes.UnsignedByte"
                initialValue="99"/>

      <Structure name="packetResolution" initialValue = "2">
```

```
            <Field name="resolution"
            type="org.web3d.vrtp.datatypes.UnsignedByte"
            initialValue="2"/>

</Structure>

<Structure name="Position">

      <Field name="x"
             type="org.web3d.vrtp.datatypes.DoublePrecision"
             initialValue="0.0"/>

      <Field name="y"
             type="org.web3d.vrtp.datatypes.DoublePrecision"
             initialValue="0.0"/>

      <Field name="z"
             type="org.web3d.vrtp.datatypes.DoublePrecision"
             initialValue="0.0"/>

</Structure>

<Structure name ="posRefTime">

      <Field name="posTime"
            type="org.web3d.vrtp.datatypes.LongInteger"
            initialValue="0"/>

</Structure>

<Structure name ="orRefTime">

      <Field name="orTime"
            type="org.web3d.vrtp.datatypes.LongInteger"
            initialValue="0"/>

</Structure>

<Structure name="orientation">

      <Field name="or1"
             type="org.web3d.vrtp.datatypes.DoublePrecision"
             initialValue="0.0"/>

      <Field name="or2"
             type="org.web3d.vrtp.datatypes.DoublePrecision"
             initialValue="0.0"/>

      <Field name="or3"
             type="org.web3d.vrtp.datatypes.DoublePrecision"
             initialValue="0.0"/>

      <Field name="or4"
             type="org.web3d.vrtp.datatypes.DoublePrecision"
             initialValue="0.0"/>

</Structure>

<Structure name="linearVelocity">

      <Field name="x"
             type="org.web3d.vrtp.datatypes.DoublePrecision"
             initialValue="0.0"/>
```

```
      <Field name="y"
             type="org.web3d.vrtp.datatypes.DoublePrecision"
             initialValue="0.0"/>

      <Field name="z"
             type="org.web3d.vrtp.datatypes.DoublePrecision"
             initialValue="0.0"/>

</Structure>

<Structure name="angularVelocity">

      <Field name="x"
             type="org.web3d.vrtp.datatypes.DoublePrecision"
             initialValue="0.0"/>

      <Field name="y"
             type="org.web3d.vrtp.datatypes.DoublePrecision"
             initialValue="0.0"/>

      <Field name="z"
             type="org.web3d.vrtp.datatypes.DoublePrecision"
             initialValue="0.0"/>

</Structure>

</ADU>

</Protocol>
```

D. DBP INERTIA LOW RESOLUTION SOURCE DOCUMENT

```
<?xml version="1.0"?>

<!DOCTYPE Protocol SYSTEM "./DynamicBehaviorProtocol.dtd">

<Protocol name="INERTIAL_PACKET"
          markerPosition="0"
          markerType="org.web3d.vrtp.datatypes.UnsignedByte"
          clientStub="http://some.url.com/baz">

<Type name="org.web3d.vrtp.datatypes.SinglePrecision"
value="file://org.web3d.vrtp.datatypes.SinglePrecision"></Type>
<Type name="org.web3d.vrtp.datatypes.UnsignedByte"
value="file://org.web3d.vrtp.datatypes.UnsignedByte"></Type>
<Type name="org.web3d.vrtp.datatypes.SignedInteger"
value="file://org.web3d.vrtp.datatypes.SignedInteger"></Type>
<Type name="org.web3d.vrtp.datatypes.UnsignedInteger"
value="file://org.web3d.vrtp.datatypes.UnsignedInteger"></Type>

<Channel multicastAddress="225.4.3.122"
                multicastPort = "1717" />

<PacketHeader name="RTP">

  <Field name="sequence"
         type="org.web3d.vrtp.datatypes.SignedInteger"
         initialValue="0"/>

  <Field name="timestamp"
         type="org.web3d.vrtp.datatypes.SignedInteger"
         initialValue="0"/>
```

```
</PacketHeader>

<ADU name="IPPPDU" markerValue="99">

  <Field name="markerValue"
         type="org.web3d.vrtp.datatypes.UnsignedByte"
         initialValue="99"/>

<Structure name="packetResolution" initialValue = "1">

          <Field name="resolution"
          type="org.web3d.vrtp.datatypes.UnsignedByte"
          initialValue="1"/>

</Structure>

<Structure name="Position">

      <Field name="x"
             type="org.web3d.vrtp.datatypes.SinglePrecision"
             initialValue="0.0"/>

      <Field name="y"
             type="org.web3d.vrtp.datatypes.SinglePrecision"
             initialValue="0.0"/>

      <Field name="z"
             type="org.web3d.vrtp.datatypes.SinglePrecision"
             initialValue="0.0"/>

</Structure>

<Structure name ="posRefTime">

      <Field name="posTime"
            type="org.web3d.vrtp.datatypes.UnsignedInteger"
            initialValue="0"/>

</Structure>

<Structure name ="orRefTime">
      <Field name="orTime"
            type="org.web3d.vrtp.datatypes.UnsignedInteger"
            initialValue="0"/>

</Structure>

<Structure name="orientation">

      <Field name="or1"
             type="org.web3d.vrtp.datatypes.SinglePrecision"
             initialValue="0.0"/>

      <Field name="or2"
             type="org.web3d.vrtp.datatypes.SinglePrecision"
             initialValue="0.0"/>

      <Field name="or3"
             type="org.web3d.vrtp.datatypes.SinglePrecision"
             initialValue="0.0"/>

      <Field name="or4"
```

```
            type="org.web3d.vrtp.datatypes.SinglePrecision"
            initialValue="0.0"/>

    </Structure>

    <Structure name="linearVelocity">

          <Field name="x"
                 type="org.web3d.vrtp.datatypes.SinglePrecision"
                 initialValue="0.0"/>

          <Field name="y"
                 type="org.web3d.vrtp.datatypes.SinglePrecision"
                 initialValue="0.0"/>

          <Field name="z"
                 type="org.web3d.vrtp.datatypes.SinglePrecision"
                 initialValue="0.0"/>

    </Structure>

    <Structure name="angularVelocity">

          <Field name="x"
                 type="org.web3d.vrtp.datatypes.SinglePrecision"
                 initialValue="0.0"/>

          <Field name="y"
                 type="org.web3d.vrtp.datatypes.SinglePrecision"
                 initialValue="0.0"/>

          <Field name="z"
                 type="org.web3d.vrtp.datatypes.SinglePrecision"
                 initialValue="0.0"/>

    </Structure>

    </ADU>

    </Protocol>
```

E. SUMMARY

The source code and source documents presented in this appendix work together to provide both a high and low resolution capability for the DBP inertia protocol.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C

A. INTRODUCTION

This appendix provides the Java and XML source code for the DBP Animation protocol.

B. DBP ANIMATION JAVA SOURCE CODE

```
package org.npsnet.v.test.dabp.dbpfoundation.dbputil;
import java.io.*;
import java.net.*;
import java.util.*;
import java.util.Vector.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import java.lang.*;

import org.npsnet.v.sys.*;
import org.npsnet.v.net.*;
import org.npsnet.v.test.dabp.dbpfoundation.*;
import org.npsnet.v.test.dabp.dbpfoundation.dbputil.*;
import org.web3d.vrtp.dabp.*;
import org.web3d.vrtp.datatypes.*;

/**
 * A protocol for transmitting information about simple, independent
 * animations with integer identifiers.
 *
 * @author Bill Fischer
 */

public class DBPAnimationProtocol extends org.npsnet.v.sys.Protocol
implements    Serializable
{
    /**
     * The class's version string.
     */
    private static final String classVersion = "$Revision: 1.3 $";

    /**
     * The known location of this class.
     */
    private static final String classCodebaseString =
"http://homer.cs.nps.navy.mil/remote/";

    /**
     * The singleton instance of this class.
     */
    private static DBPAnimationProtocol animationProtocol;

    /**
     * The known location of this class.
     */
    private static final String PROTO_URL =
"http://homer.cs.nps.navy.mil/remote/";

    /**
     * The URL to the XML file describing the packet layout
```

```
             */
            String hiXmlFile = new
        String("c:/npsnetV/org/npsnet/v/test/dabp/AnimationProtocolPacket.xml");

            /**
             * The URL to the XML file describing the High Resolution packet
layout
             */
            String loXmlFile = new
        String("c:/npsnetV/org/npsnet/v/test/dabp/AnimationProtocolPacket.xml");

            /**
             * DABP protocols for high resolution
             */
            private org.web3d.vrtp.dabp.Protocol hiResDabpProtocol = new
        org.web3d.vrtp.dabp.Protocol(hiXmlFile);

            /**
             * DABP protocols for high resolution
             */
            private org.web3d.vrtp.dabp.Protocol loResDabpProtocol = new
        org.web3d.vrtp.dabp.Protocol(loXmlFile);

            /**
             * Returns this class's version string.
             */
            public String getClassVersion() {
                return classVersion;
            }

            /**
             * Returns this class's codebase.
             */
            public URL getClassCodebase() {
                try { return new URL(classCodebaseString); }
                catch(MalformedURLException e) { return null; }
            }

            /**
             * Returns the singleton instance of <code>AnimationProtocol</code>.
             */
            public static DBPAnimationProtocol getInstance() {
                if(animationProtocol!=null) return animationProtocol;
                else {
                    try {
               org.npsnet.v.sys.Protocol tmpProt =
        getSingleton("org.npsnet.v.test.dabp.dbpfoundation.dbputil.DBPAnimationPr
otoc   ol",classVersion);

                        if(tmpProt == null)
                            animationProtocol = new DBPAnimationProtocol();
                        else
                            animationProtocol = (DBPAnimationProtocol)tmpProt;
                    }
                    catch(IDServerException ise) {
                        System.out.println(ise);
                        System.exit(1);
                    }
                    return animationProtocol;
                }
            }

            /**
```

```
             * Constructor; takes a few parameters, which allow us
             * to operate in the world.<p>
             *
             * Note that this constructor is for use with new EntityMasters--
             * We create a new entity and send out information on it. The
             * creation of the entity registers a new entity ID and new
protocolIDs
             * to go along with it.<p>
             *
             * @param pEntity entity that we update
             */
            public DBPAnimationProtocol() throws IDServerException
            {
                // A new instance requires a new ID
                   super("DBPAnimationProtocol", true);
            }

            /**
             * constructor. Note that this constructor is for use with
             * things we have discovered from the net. we get in a new
             * message with an existing ID. That means we don't have to
             * look up the ID on the IDServer.<p>
             *
             * @param pID id of the protocol, gleaned from network traffic
             */
            public DBPAnimationProtocol(long pID) throws IDServerException
            {
                super(pID, IDCache.getIDCache().getNameForID(pID));
            }

            /**
             * This receives a packet from the entity and serializes it, ie,
changes
             * all the data fields into a byte array that we can send across the
             * network. ADU is some nominal superclass from DABP that subsumes
all
             * packets.
             */
            public byte[] serializeADU(Object pPacketData)
            {
              byte                  serializedData[] = null;
              ADUData               animationADU = null;
              animationADU = (ADUData)pPacketData;

              ByteArrayOutputStream oStream = new ByteArrayOutputStream();
              DataOutputStream dos = new DataOutputStream(oStream);

              animationADU.serialize(dos);
              serializedData = oStream.toByteArray();

              return serializedData;
            }

            /**
             * Receives a packet from the EntityDispatcher (or whatever.) The
             * data we get in is just an array of bytes--we have to interpret
those
             * bytes so that we can do something with them. This consists of two
             * things: determing syntax, so we can retrieve the field named
             * "position.x"; and determining semantics, so we know that
position.x
             * refers to the location of the entity in 3-space. We instantiate
             * an ADUData packet, lay it out in the format described by the
```

```
           * protocol's XML document, and fill in the packet's fields from the
           * incoming byte stream data.
           */
          public void receivePacket(byte[] pData, NetworkAware entity)
          {
          ByteArrayInputStream bais        = new ByteArrayInputStream(pData);

          // Temporary workaround; prevents EntityMasters from being
          // modified remotely
          if(entity instanceof EntityMaster) return;
          if(entity instanceof DBPAnimatedMaster) return;

          // We only get packets of one type, so we don't need to check
          // on format beforehand (famous last words).

           ADUData packet = null;
           packet = hiResDabpProtocol.binaryDataToADU(pData); //binaryDataToADU
          // some temporary objects & variables
          boolean animIsLooping = false;
          long animStartTime = 0;
          boolean lowResPacket = false;

          try
          {
          if( ((UnsignedByte)(packet.get("markerValue"))).byteValue() == 96)
            {

if(((UnsignedByte)(packet.get("packetResolution.resolution"))).byteValue() ==
1)
            {  packet = loResDabpProtocol.binaryDataToADU(pData);
//binaryDataToADU
            }// end if

            // some temporary variables
            byte tempLooping = 0;

            tempLooping =
      (byte)(((PrimitiveNumber)packet.get("animationLooping.looping")).byteValu
e());

            PrimitiveNumber startTimePrim =
      ((PrimitiveNumber)packet.get("animStartTime.time"));

            if(startTimePrim instanceof LongInteger)
              animStartTime = (long)(startTimePrim.longValue());

            else if(startTimePrim instanceof UnsignedInteger)
            {
              long sysTime = System.currentTimeMillis();
              long timeMask = 0xffffffff00000000l;
              sysTime = sysTime & timeMask;
              long tempInt = startTimePrim.longValue();
              animStartTime = (sysTime | tempInt);
            }//end else if
            animIsLooping = tempLooping == 1 ? true : false;
      //      System.out.println("Receive side --
AnimationProtocol:receivePacket():packet ==     "+ packet);

          double one = 1;
               ((DBPAnimated)entity).setAnimation((int)(entity.getID()),
animIsLooping,             animStartTime, one);
          }// end if
          else
```

```
            { System.err.println("Wrong marker type (not 96)");
            }// end else
            }// end try
            catch(FieldNotFoundException fnfe)
              { System.err.println("Field not found! " + fnfe);
              }// end catch
            return;
            }

            /**
             * Invoked when an entity is modified.
             */
            public void entityChanged(Entity e, Object info)
            {
            if(e instanceof DBPAnimatedGhost) return;

              ADUData animatedADU = null;
              animatedADU = new ADUData(new
          ADU(hiResDabpProtocol.getSchemaNamed("ANPPPDU")));

              if(resolution == 1)
              { animatedADU = new ADUData(new
          ADU(loResDabpProtocol.getSchemaNamed("ANPPPDU")));
              }// end if

              boolean animIsLooping = ((DBPAnimated)e).getAnimIsLooping();
              long animStartTime = ((DBPAnimated)e).getAnimStartTime();
              byte animLooping = (byte)(animIsLooping == true ? 1 : 0);

              try
              {
              // load the ADUData packet
                  animatedADU.put("packetResolution.resolution", new
          UnsignedByte(resolution));

                  String tempString = new String("animationLooping.looping");

                  if(animatedADU.get(tempString) instanceof UnsignedByte)
                    animatedADU.put(tempString, new UnsignedByte(animLooping));
                  else if(animatedADU.get(tempString) instanceof UnsignedShort)
                    animatedADU.put(tempString, new
    UnsignedShort((short)animLooping));

                  tempString = new String("animStartTime.time");
                  if(animatedADU.get(tempString) instanceof LongInteger)
                    animatedADU.put(tempString, new LongInteger(animStartTime));
                  else if(animatedADU.get(tempString) instanceof UnsignedInteger)
                  {
                    long sysTime = System.currentTimeMillis();
                    sysTime = sysTime & 0x00000000ffffffffl;
                    int tempInt = (int)sysTime;
                    UnsignedInteger usIntTime = new UnsignedInteger(tempInt);
                    animatedADU.put(tempString, usIntTime);
                  }// end else if
              }// end try
              catch(FieldTypeException fte)
              {
                System.err.println("Field Type not found: " + fte);
              }// end catch
              catch(FieldNotFoundException fnfe)
              {
                System.err.println("Field not found: " + fnfe);
              }// end catch
```

```
      //      System.out.println("Send side ---
AnimationProtocol:entityChanged():animatedADU == \n" + animatedADU);

            // Serialize and send animation data for the entity
            byte[] data = serializeADU(animatedADU);

            Set distroSet = AOIM.getAOIM().getDistributionSet(e, this);

            EntityDispatcher.getEntityDispatcher().sendPacket(this,
                                                              e.getID(),
                                                              distroSet,
                                                              data);
         }
      };
```

C. DBP ANIMATION HIGH RESOLUTION SOURCE DOCUMENT

```
      <?xml version="1.0"?>

      <!DOCTYPE Protocol SYSTEM "./DynamicBehaviorProtocol.dtd">

      <Protocol name="ANIMATION_STATE"
                markerPosition="0"
                markerType="org.web3d.vrtp.datatypes.UnsignedByte"
                clientStub="http://some.url.com/baz">

      <Type name="org.web3d.vrtp.datatypes.DoublePrecision"
      value="file://org.web3d.vrtp.datatypes.DoublePrecision"></Type>

      <Type name="org.web3d.vrtp.datatypes.LongInteger"
      value="file://org.web3d.vrtp.datatypes.LongInteger"></Type>

      <Type name="org.web3d.vrtp.datatypes.UnsignedByte"
      value="file://org.web3d.vrtp.datatypes.UnsignedByte"></Type>

      <Type name="org.web3d.vrtp.datatypes.SignedInteger"
      value="file://org.web3d.vrtp.datatypes.SignedInteger"></Type>

      <!-- <Channel multicastAddress="225.7.8.30"
                     multicastPort = "4070" /> -->

      <Channel multicastAddress="225.9.9.127"
                     multicastPort = "1919" />

      <PacketHeader name="RTP">

        <Field name="sequence"
               type="org.web3d.vrtp.datatypes.SignedInteger"
               initialValue="0"/>

        <Field name="timestamp"
               type="org.web3d.vrtp.datatypes.SignedInteger"
               initialValue="0"/>

      </PacketHeader>

      <ADU name="ANPPPDU" markerValue="96">

        <Field name="markerValue"
                type="org.web3d.vrtp.datatypes.UnsignedByte"
                initialValue="96"/>
```

```
<Structure name="packetResolution" initialValue = "2">

          <Field name="resolution"
          type="org.web3d.vrtp.datatypes.UnsignedByte"
          initialValue="2"/>

</Structure>

<Structure name ="animationLooping">

      <Field name="looping"
            type="org.web3d.vrtp.datatypes.UnsignedByte"
            initialValue="0"/>

</Structure>

<Structure name ="animStartTime">

      <Field name="time"
            type="org.web3d.vrtp.datatypes.LongInteger"
            initialValue="0"/>

</Structure>

</ADU>

</Protocol>
```

D. DBP ANIMATION LOW RESOLUTION SOURCE DOCUMENT

```
<?xml version="1.0"?>

<!DOCTYPE Protocol SYSTEM "./DynamicBehaviorProtocol.dtd">

<Protocol name="ANIMATION_STATE"
          markerPosition="0"
          markerType="org.web3d.vrtp.datatypes.UnsignedByte"
          clientStub="http://some.url.com/baz">

<Type name="org.web3d.vrtp.datatypes.DoublePrecision"
value="file://org.web3d.vrtp.datatypes.DoublePrecision"></Type>

<Type name="org.web3d.vrtp.datatypes.LongInteger"
value="file://org.web3d.vrtp.datatypes.LongInteger"></Type>

<Type name="org.web3d.vrtp.datatypes.UnsignedByte"
value="file://org.web3d.vrtp.datatypes.UnsignedByte"></Type>

<Type name="org.web3d.vrtp.datatypes.SignedInteger"
value="file://org.web3d.vrtp.datatypes.SignedInteger"></Type>

<Type name="org.web3d.vrtp.datatypes.UnsignedInteger"
value="file://org.web3d.vrtp.datatypes.UnsignedInteger"></Type>

<!-- <Channel multicastAddress="225.7.8.30"
                multicastPort = "4070" /> -->

<Channel multicastAddress="225.9.9.127"
                multicastPort = "1919" />

<PacketHeader name="RTP">
```

```
  <Field name="sequence"
         type="org.web3d.vrtp.datatypes.SignedInteger"
         initialValue="0"/>

  <Field name="timestamp"
         type="org.web3d.vrtp.datatypes.SignedInteger"
         initialValue="0"/>

</PacketHeader>


<ADU name="ANPPPDU" markerValue="96">

  <Field name="markerValue"
          type="org.web3d.vrtp.datatypes.UnsignedByte"
          initialValue="96"/>

<Structure name="packetResolution" initialValue = "1">

          <Field name="resolution"
          type="org.web3d.vrtp.datatypes.UnsignedByte"
          initialValue="1"/>

</Structure>

<Structure name ="animationLooping">

      <Field name="looping"
            type="org.web3d.vrtp.datatypes.UnsignedByte"
            initialValue="0"/>

</Structure>

<Structure name ="animStartTime">

      <Field name="time"
            type="org.web3d.vrtp.datatypes.UnsignedInteger"
            initialValue="0"/>

</Structure>

</ADU>

</Protocol>
```

E. SUMMARY

The source code and source documents presented in this appendix work together to provide both a high and low resolution capability for the DBP animation protocol.

APPENDIX D

A. INTRODUCTION

This appendix provides the Java and XML source code for the DBP acceleration protocol.

B. DBP ACCELERATION JAVA SOURCE CODE

```
package org.npsnet.v.test.dabp.dbpfoundation.dbputil;

import java.io.*;
import java.net.*;
import java.util.*;
import java.util.Vector.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import java.lang.*;

import org.npsnet.v.sys.*;
import org.npsnet.v.net.*;
import org.npsnet.v.test.dabp.dbpfoundation.*;
import org.npsnet.v.test.dabp.dbpfoundation.dbputil.*;
import org.web3d.vrtp.dabp.*;
import org.web3d.vrtp.datatypes.*;

/**
 * A protocol that passes information about linear and angular
 * acceleration.
 *
 * @author Bill Fischer
 */

public class DBPAccelerationProtocol extends org.npsnet.v.sys.Protocol
implements Serializable
{
    /**
     * The class's version string.
     */
    private static final String classVersion = "$Revision: 1.2 $";

    /**
     * The known location of this class.
     */
    private static final String classCodebaseString =
"http://homer.cs.nps.navy.mil/remote/";

    /**
     * The singleton instance of this class.
     */
    private static DBPAccelerationProtocol accelerationProtocol;

    /**
     * The known location of this class.
     */
    private static final String PROTO_URL =
"http://homer.cs.nps.navy.mil/remote/";

    /**
```

```
            * The URL to the XML file describing the High Resolution packet
layout
            */
           String hiXmlFile = new
       String("c:/npsnetV/org/npsnet/v/test/dabp/AccelerationProtocolPacket.xml"
);

           /**
            * The URL to the XML file describing the High Resolution packet
layout
            */
           String loXmlFile = new
       String("c:/npsnetV/org/npsnet/v/test/dabp/AccelerationProtocolPacketLoRes
.xml");

           /**
            * DABP protocols for high resolution
            */
           private org.web3d.vrtp.dabp.Protocol hiResDabpProtocol = new
       org.web3d.vrtp.dabp.Protocol(hiXmlFile);

           /**
            * DABP protocols for high resolution
            */
           private org.web3d.vrtp.dabp.Protocol loResDabpProtocol = new
       org.web3d.vrtp.dabp.Protocol(loXmlFile);

           /**
            * Returns this class's version string.
            */
           public String getClassVersion() {
               return classVersion;
           }

           /**
            * Returns this class's codebase.
            */
           public URL getClassCodebase() {
               try { return new URL(classCodebaseString); }
               catch(MalformedURLException e) { return null; }
           }

           /**
            * Returns the singleton instance of
<code>AccelerationProtocol</code>.
            */
           public static DBPAccelerationProtocol getInstance() {
               if(accelerationProtocol!=null) return accelerationProtocol;
               else {
                   try {
                       org.npsnet.v.sys.Protocol tmpProt =
getSingleton("org.npsnet.v.test.dabp.dbpfoundation.dbputil.DBPAccelerationProto
col",classVersion);

                       if(tmpProt == null)
                           accelerationProtocol = new DBPAccelerationProtocol();
                       else
                           accelerationProtocol =
(DBPAccelerationProtocol)tmpProt;
                   }
                   catch(IDServerException ise) {
                       System.out.println(ise);
                       System.exit(1);
```

```
                }
                return accelerationProtocol;
            }
        }

        /**
         * Constructor; takes a few parameters, which allow us
         * to operate in the world.<p>
         * Note that this constructor is for use with new EntityMasters--
         * We create a new entity and send out information on it. The
         * creation of the entity registers a new entity ID and new
protocolIDs
         * to go along with it.<p>
         *
         * @param pEntity entity that we update
         */
        public DBPAccelerationProtocol() throws IDServerException
        {
            // A new instance requires a new ID
               super("DBPAccelerationProtocol", true);
        }

        /**
         * constructor. Note that this constructor is for use with
         * things we have discovered from the net. we get in a new
         * message with an existing ID. That means we don't have to
         * look up the ID on the IDServer.<p>
         *
         * @param pID id of the protocol, gleaned from network traffic
         */
        public DBPAccelerationProtocol(long pID) throws IDServerException
        {
            super(pID, IDCache.getIDCache().getNameForID(pID));
        }

        /**
         * This receives a packet from the entity and serializes it, ie,
changes
         * all the data fields into a byte array that we can send across the
         * network. ADU is some nominal superclass from DABP that subsumes
all
         * packets.
         */
        public byte[] serializeADU(Object pPacketData)
        {
          byte                  serializedData[] = null;
          ADUData               accelerationADU = null;
          accelerationADU = (ADUData)pPacketData;

          ByteArrayOutputStream oStream = new ByteArrayOutputStream();
          DataOutputStream dos = new DataOutputStream(oStream);

          accelerationADU.serialize(dos);
          serializedData = oStream.toByteArray();

          return serializedData;
        }

        /**
         * Receives a packet from the EntityDispatcher (or whatever.) The
         * data we get in is just an array of bytes--we have to interpret
those
         * bytes so that we can do something with them. This consists of two
```

```
            * things: determing syntax, so we can retrieve the field named
            * "position.x"; and determining semantics, so we know that
position.x
            * refers to the location of the entity in 3-space. We instantiate
            * an ADUData packet, lay it out in the format described by the
            * protocol's XML document, and fill in the packet's fields from the
            * incoming byte stream data.
            */
           public void receivePacket(byte[] pData, NetworkAware entity)
           {
           ByteArrayInputStream bais        = new ByteArrayInputStream(pData);

           // Temporary workaround; prevents EntityMasters from being
           // modified remotely
           if(entity instanceof EntityMaster) return;
           if(entity instanceof DBPAccelerableMaster) return;

           // We only get packets of one type, so we don't need to check
           // on format beforehand (famous last words).

            ADUData packet = null;
            packet = hiResDabpProtocol.binaryDataToADU(pData); //binaryDataToADU

           // some temporary objects & variables
           Vector3d linearAcceleration = null;
           Vector3d angularAcceleration = null;

           try
           {
           if( ((UnsignedByte)(packet.get("markerValue"))).byteValue() == 97)
             {

if(((UnsignedByte)(packet.get("packetResolution.resolution"))).byteValue() ==
1)
             {  packet = loResDabpProtocol.binaryDataToADU(pData);
//binaryDataToADU
             }// end if

             // some temporary variables
             double linX, linY, linZ, angX, angY, angZ;

             // fill the blank ADUData packet with the corresponding fields from
the
             //    incoming byte stream.  Try separately in case later we decide
to
             //    only send a partial packet (eg linear and not angular)

             try{
               linX =
(double)(((PrimitiveNumber)packet.get("linearAcceleration.lax")).doubleValue())
;
               linY =
(double)(((PrimitiveNumber)packet.get("linearAcceleration.lay")).doubleValue())
;
               linZ =
(double)(((PrimitiveNumber)packet.get("linearAcceleration.laz")).doubleValue())
;
               linearAcceleration = new Vector3d(linX, linY, linZ);
             }// end try
             catch(FieldNotFoundException fnfe)
             { System.err.println("Field not found! " + fnfe);
             }// end catch
```

```
            try
            { angX =
(double)(((PrimitiveNumber)packet.get("angularAcceleration.aax")).doubleValue()
);
              angY =
(double)(((PrimitiveNumber)packet.get("angularAcceleration.aay")).doubleValue()
);
              angZ =
(double)(((PrimitiveNumber)packet.get("angularAcceleration.aaz")).doubleValue()
);
              angularAcceleration = new Vector3d(angX, angY, angZ);
            }// end try
            catch(FieldNotFoundException fnfe)
            { System.err.println("Field not found! " + fnfe);
            }// end catch

      //      System.out.println("Receive side --
TransformProtocol:receivePacket():packet ==     "+ packet);

            if(linearAcceleration != null)
            {
((DBPAccelerable)entity).setLinearAcceleration(linearAcceleration);
            }// end if

            if(angularAcceleration != null)
            {
((DBPAccelerable)entity).setAngularAcceleration(angularAcceleration);
            }// end if

          }// end if
          else
          { System.err.println("Wrong marker type (not 97)");
          }// end else

          }// end try
          catch(FieldNotFoundException fnfe)
            { System.err.println("Field not found! " + fnfe);
            }// end catch

          return;
          }

          /**
           * Invoked when an entity is modified.  The entity's inertia
information
           * is loaded into an ADUData packet, which is then serialized and
passed
           * to the Entity Master.
           */
          public void entityChanged(Entity e, Object info)
          {
          if(e instanceof DBPAccelerableGhost) return;

            ADUData accelerationADU = null;
            accelerationADU = new ADUData(new
      ADU(hiResDabpProtocol.getSchemaNamed("ACPPPDU")));

            if(resolution == 1)
            { accelerationADU = new ADUData(new
      ADU(loResDabpProtocol.getSchemaNamed("ACPPPDU")));
            }// end if

            // linear acceleration
```

```
            double[] linearAccelerationArray = new double[3];

            Vector3d linearAcceleration = new Vector3d();
            ((DBPAccelerable)e).getLinearAcceleration(linearAcceleration);
            linearAcceleration.get(linearAccelerationArray);

            double linX = linearAccelerationArray[0];
            double linY = linearAccelerationArray[1];
            double linZ = linearAccelerationArray[2];

            // angular acceleration
            double[] angularAccelerationArray = new double[3];

            Vector3d angularAcceleration = new Vector3d();
            ((DBPAccelerable)e).getAngularAcceleration(angularAcceleration);
            angularAcceleration.get(angularAccelerationArray);

            double angX = angularAccelerationArray[0];
            double angY = angularAccelerationArray[1];
            double angZ = angularAccelerationArray[2];
            try
            {
            // load the ADUData packet
              accelerationADU.put("packetResolution.resolution", new
      UnsignedByte(resolution));
            }// end try
            catch(FieldTypeException fte)
            { System.err.println("Field Type not found: " + fte);
            }// end catch
            catch(FieldNotFoundException fnfe)
            { System.err.println("Field not found: " + fnfe);
            }// end catch

            String tempString = new String("linearAcceleration.lax");
            // doing this in pieces in case we later want to send one without
the other
            try
            {
              if(accelerationADU.get(tempString) instanceof DoublePrecision)
                accelerationADU.put(tempString, new DoublePrecision(linX));
              else if(accelerationADU.get(tempString) instanceof
SinglePrecision)
                accelerationADU.put(tempString, new
SinglePrecision((float)linX));

              tempString = new String("linearAcceleration.lay");
              if(accelerationADU.get(tempString) instanceof DoublePrecision)
                accelerationADU.put(tempString, new DoublePrecision(linY));
              else if(accelerationADU.get(tempString) instanceof
SinglePrecision)
                accelerationADU.put(tempString, new
SinglePrecision((float)linY));

              tempString = new String("linearAcceleration.laz");
              if(accelerationADU.get(tempString) instanceof DoublePrecision)
                accelerationADU.put(tempString, new DoublePrecision(linZ));
              else if(accelerationADU.get(tempString) instanceof
SinglePrecision)
                accelerationADU.put(tempString, new
SinglePrecision((float)linZ));
              }// end try
            catch(FieldTypeException fte)
            { System.err.println("Field Type not found: " + fte);
```

```
            }// end catch
              catch(FieldNotFoundException fnfe)
              { System.err.println("Field not found: " + fnfe);
              }// end catch

              tempString = new String("angularAcceleration.aax");
            try{
              if(accelerationADU.get(tempString) instanceof DoublePrecision)
                accelerationADU.put(tempString, new DoublePrecision(angX));
              else if(accelerationADU.get(tempString) instanceof
SinglePrecision)
                accelerationADU.put(tempString, new
SinglePrecision((float)angX));

              tempString = new String("angularAcceleration.aay");
              if(accelerationADU.get(tempString) instanceof DoublePrecision)
                accelerationADU.put(tempString, new DoublePrecision(angY));
              else if(accelerationADU.get(tempString) instanceof
SinglePrecision)
                accelerationADU.put(tempString, new
SinglePrecision((float)angY));

              tempString = new String("angularAcceleration.aaz");
              if(accelerationADU.get(tempString) instanceof DoublePrecision)
                accelerationADU.put(tempString, new DoublePrecision(angZ));
              else if(accelerationADU.get(tempString) instanceof
SinglePrecision)
                accelerationADU.put(tempString, new
SinglePrecision((float)angZ));
            }// end try
            catch(FieldTypeException fte)
            { System.err.println("Field Type not found: " + fte);
            }// end catch
            catch(FieldNotFoundException fnfe)
            { System.err.println("Field not found: " + fnfe);
            }// end catch



      //      System.out.println("Send side ---
      AccelerationProtocol:entityChanged():accelerationADU == \n" +
      accelerationADU);//*************888888888888888888

            // Serialize and send inertial data for the entity
            // Serialize and send transform data for the entity
            byte[] data = serializeADU(accelerationADU);

            Set distroSet = AOIM.getAOIM().getDistributionSet(e, this);

            EntityDispatcher.getEntityDispatcher().sendPacket(this,
                                                              e.getID(),
                                                              distroSet,
                                                              data);
        }
      };
```

C. DBP ACCELERATION HIGH RESOLUTION SOURCE DOCUMENT

```
      <?xml version="1.0"?>

      <!DOCTYPE Protocol SYSTEM "./DynamicBehaviorProtocol.dtd">

      <Protocol name="ACCELERATION_STATE"
```

```
          markerPosition="0"
          markerType="org.web3d.vrtp.datatypes.UnsignedByte"
          clientStub="http://some.url.com/baz">


<Type name="org.web3d.vrtp.datatypes.DoublePrecision"
value="file://org.web3d.vrtp.datatypes.DoublePrecision"></Type>

<Type name="org.web3d.vrtp.datatypes.LongInteger"
value="file://org.web3d.vrtp.datatypes.LongInteger"></Type>

<Type name="org.web3d.vrtp.datatypes.UnsignedByte"
value="file://org.web3d.vrtp.datatypes.UnsignedByte"></Type>

<Type name="org.web3d.vrtp.datatypes.SignedInteger"
value="file://org.web3d.vrtp.datatypes.SignedInteger"></Type>

<Channel multicastAddress="225.2.4.122"
               multicastPort = "1818" />

<PacketHeader name="RTP">

  <Field name="sequence"
         type="org.web3d.vrtp.datatypes.SignedInteger"
         initialValue="0"/>

  <Field name="timestamp"
         type="org.web3d.vrtp.datatypes.SignedInteger"
         initialValue="0"/>

</PacketHeader>

<ADU name="ACPPPDU" markerValue="97">

<!-- <Field name="markerValue"
          type="org.web3d.vrtp.datatypes.UnsignedByte"
          initialValue="97"/> -->

<Structure name="packetResolution" initialValue = "2">

          <Field name="resolution"
          type="org.web3d.vrtp.datatypes.UnsignedByte"
          initialValue="2"/>

</Structure>

<Structure name="linearAcceleration">

      <Field name="lax"
             type="org.web3d.vrtp.datatypes.DoublePrecision"
             initialValue="0.0"/>

      <Field name="lay"
             type="org.web3d.vrtp.datatypes.DoublePrecision"
             initialValue="0.0"/>

      <Field name="laz"
             type="org.web3d.vrtp.datatypes.DoublePrecision"
             initialValue="0.0"/>

</Structure>

<Structure name="angularAcceleration">
```

```
            <Field name="aax"
                   type="org.web3d.vrtp.datatypes.DoublePrecision"
                   initialValue="0.0"/>

            <Field name="aay"
                   type="org.web3d.vrtp.datatypes.DoublePrecision"
                   initialValue="0.0"/>

            <Field name="aaz"
                   type="org.web3d.vrtp.datatypes.DoublePrecision"
                   initialValue="0.0"/>

      </Structure>

      </ADU>

      </Protocol>
```

D. DBP ACCELERATION LOW RESOLUTION SOURCE DOCUMENT

```
      <?xml version="1.0"?>

      <!DOCTYPE Protocol SYSTEM "./DynamicBehaviorProtocol.dtd">

      <Protocol name="ACCELERATION_STATE"
                markerPosition="0"
                markerType="org.web3d.vrtp.datatypes.UnsignedByte"
                clientStub="http://some.url.com/baz">

      <Type name="org.web3d.vrtp.datatypes.SinglePrecision"
value="file://org.web3d.vrtp.datatypes.SinglePrecision"></Type>

      <Type name="org.web3d.vrtp.datatypes.LongInteger"
value="file://org.web3d.vrtp.datatypes.LongInteger"></Type>

      <Type name="org.web3d.vrtp.datatypes.UnsignedByte"
value="file://org.web3d.vrtp.datatypes.UnsignedByte"></Type>

      <Type name="org.web3d.vrtp.datatypes.SignedInteger"
value="file://org.web3d.vrtp.datatypes.SignedInteger"></Type>

      <Channel multicastAddress="225.2.4.122"
                     multicastPort = "1818" />

      <PacketHeader name="RTP">

        <Field name="sequence"
               type="org.web3d.vrtp.datatypes.SignedInteger"
               initialValue="0"/>

        <Field name="timestamp"
               type="org.web3d.vrtp.datatypes.SignedInteger"
               initialValue="0"/>

      </PacketHeader>

      <ADU name="ACPPPDU" markerValue="97">

      <!-- <Field name="markerValue"
                type="org.web3d.vrtp.datatypes.UnsignedByte"
                initialValue="97"/> -->
```

```
<Structure name="packetResolution" initialValue = "2">

          <Field name="resolution"
          type="org.web3d.vrtp.datatypes.UnsignedByte"
          initialValue="1"/>

</Structure>

<Structure name="linearAcceleration">

      <Field name="lax"
             type="org.web3d.vrtp.datatypes.SinglePrecision"
             initialValue="0.0"/>

      <Field name="lay"
             type="org.web3d.vrtp.datatypes.SinglePrecision"
             initialValue="0.0"/>

      <Field name="laz"
             type="org.web3d.vrtp.datatypes.SinglePrecision"
             initialValue="0.0"/>

</Structure>

<Structure name="angularAcceleration">

      <Field name="aax"
             type="org.web3d.vrtp.datatypes.SinglePrecision"
             initialValue="0.0"/>

      <Field name="aay"
             type="org.web3d.vrtp.datatypes.SinglePrecision"
             initialValue="0.0"/>

      <Field name="aaz"
             type="org.web3d.vrtp.datatypes.SinglePrecision"
             initialValue="0.0"/>

</Structure>

</ADU>

</Protocol>
```

E. SUMMARY

The source code and source documents presented in this appendix work together to provide both a high and low resolution capability for the DBP acceleration protocol.

APPENDIX E

A. INTRODUCTION

This appendix provides the Java and XML source code for the DBP articulation protocol.

B. DBP ARTICULATION JAVA SOURCE CODE

```
package org.npsnet.v.test.dabp.dbpfoundation.dbputil;

import java.io.*;
import java.net.*;
import java.util.*;
import java.util.Vector.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import java.lang.*;

import org.npsnet.v.sys.*;
import org.npsnet.v.net.*;
import org.npsnet.v.test.dabp.dbpfoundation.*;
import org.web3d.vrtp.dabp.*;
import org.web3d.vrtp.datatypes.*;

/**
 * A protocol for transmitting information about the joint positions
 * in articulated entities.  Each joint uses one protocol!
 *
 * @author Bill Fischer
 */

public class DBPArticulationProtocol extends org.npsnet.v.sys.Protocol
implements    Serializable
{
    /**
     * The class's version string.
     */
    private static final String classVersion = "$Revision: 1.2 $";

    /**
     * The known location of this class.
     */
    private static final String classCodebaseString =
"http://homer.cs.nps.navy.mil/remote/";

    /**
     * The singleton instance of this class.
     */
    private static DBPArticulationProtocol articulationProtocol;

    /**
     * The known location of this class.
     */
    private static final String PROTO_URL =
"http://homer.cs.nps.navy.mil/remote/";

    /**
```

```
             * The URL to the XML file describing the High Resolution packet
layout
             */

            String hiXmlFile = new
       String("c:/npsnetV/org/npsnet/v/test/dabp/ArticulationProtocolPacket.xml"
);

            /**
             * The URL to the XML file describing the High Resolution packet
layout
             */
            String loXmlFile = new
       String("c:/npsnetV/org/npsnet/v/test/dabp/ArticulationProtocolPacketLoRes
.xml"  );

            /**
             * DABP protocols for high resolution
             */
            private org.web3d.vrtp.dabp.Protocol hiResDabpProtocol = new
       org.web3d.vrtp.dabp.Protocol(hiXmlFile);

            /**
             * DABP protocols for high resolution
             */
            private org.web3d.vrtp.dabp.Protocol loResDabpProtocol = new
       org.web3d.vrtp.dabp.Protocol(loXmlFile);

            /**
             * The index number of the joint this protocol is for
             */
            private int jointNumber = 0;

            /**
             * Returns this class's version string.
             */
            public String getClassVersion() {
                return classVersion;
            }

            /**
             * Returns this class's codebase.
             */
            public URL getClassCodebase() {
                try { return new URL(classCodebaseString); }
                catch(MalformedURLException e) { return null; }
            }

            /**
             * Returns the singleton instance of
<code>ArticulationProtocol</code>.
             */
            public static DBPArticulationProtocol getInstance() {
                if(articulationProtocol!=null) return articulationProtocol;
                else {
                    try {
                        org.npsnet.v.sys.Protocol tmpProt =
       getSingleton("org.npsnet.v.test.dabp.dbpfoundation.dbputil.DBPArticulatio
nProto col",classVersion);

                        if(tmpProt == null)
                            articulationProtocol = new DBPArticulationProtocol();
                        else
```

```
                        articulationProtocol =
(DBPArticulationProtocol)tmpProt;
                }
                catch(IDServerException ise) {
                    System.out.println(ise);
                    System.exit(1);
                }

                return articulationProtocol;
            }
        }

        /**
         * Constructor; takes a few parameters, which allow us
         * to operate in the world.<p>
         *
         * Note that this constructor is for use with new EntityMasters--
         * We create a new entity and send out information on it. The
         * creation of the entity registers a new entity ID and new
protocolIDs
         * to go along with it.<p>
         *
         * @param pEntity entity that we update
         */
        public DBPArticulationProtocol() throws IDServerException
        {
            // A new instance requires a new ID
               super("dbpArticulationProtocol", true);
        }

        /**
         * constructor. Note that this constructor is for use with
         * things we have discovered from the net. we get in a new
         * message with an existing ID. That means we don't have to
         * look up the ID on the IDServer.<p>
         *
         * @param pID id of the protocol, gleaned from network traffic
         */

        public DBPArticulationProtocol(long pID) throws IDServerException
        {
            super(pID, IDCache.getIDCache().getNameForID(pID));
        }

      /**
       * This receives a packet from the entity and serializes it, ie,
changes
       * all the data fields into a byte array that we can send across the
       * network. ADU is some nominal superclass from DABP that subsumes all
       * packets.
       */

      public byte[] serializeADU(Object pPacketData)
      {

          byte                 serializedData[] = null;
          ADUData              articulationADU = null;
          articulationADU = (ADUData)pPacketData;

          ByteArrayOutputStream oStream = new ByteArrayOutputStream();
          DataOutputStream dos = new DataOutputStream(oStream);

          articulationADU.serialize(dos);
```

```
            serializedData = oStream.toByteArray();

            return serializedData;
        }

          /**
           * Receives a packet from the EntityDispatcher (or whatever.) The
           * data we get in is just an array of bytes--we have to interpret
those
           * bytes so that we can do something with them. This consists of two
           * things: determing syntax, so we can retrieve the field named
           * "position.x"; and determining semantics, so we know that
position.x
           * refers to the location of the entity in 3-space. We instantiate
           * an ADUData packet, lay it out in the format described by the
           * protocol's XML document, and fill in the packet's fields from the
           * incoming byte stream data.
           */

        public void receivePacket(byte[] pData, NetworkAware entity)
        {
          ByteArrayInputStream bais        = new ByteArrayInputStream(pData);

          // Temporary workaround; prevents EntityMasters from being
          // modified remotely
          if(entity instanceof EntityMaster) return;
          if(entity instanceof DBPArticulatedMaster) return;

          // We only get packets of one type, so we don't need to check
          // on format beforehand (famous last words).

           ADUData packet = null;
           packet = hiResDabpProtocol.binaryDataToADU(pData); //binaryDataToADU

          // some temporary objects & variables
          Quat4d jointOrientations = null;

          try
          {
          if( ((UnsignedByte)(packet.get("markerValue"))).byteValue() == 95)
            {

if(((UnsignedByte)(packet.get("packetResolution.resolution"))).byteValue() ==
1)
            {  packet = loResDabpProtocol.binaryDataToADU(pData);
//binaryDataToADU
            }// end if

            // some temporary variables
            double jox, joy, joz, jow;

            // fill the blank ADUData packet with the corresponding fields from
the
            //    incoming byte stream

            jox =
(double)(((PrimitiveNumber)packet.get("orientation.or1")).doubleValue());
            joy =
(double)(((PrimitiveNumber)packet.get("orientation.or2")).doubleValue());
            joz =
(double)(((PrimitiveNumber)packet.get("orientation.or3")).doubleValue());
            jow =
(double)(((PrimitiveNumber)packet.get("orientation.or4")).doubleValue());
```

```
            jointOrientations = new Quat4d(jox, joy, joz, jow);

       //      System.out.println("Receive side --
TransformProtocol:receivePacket():packet ==     "+ packet);

                ((DBPArticulated)entity).setJointOrientation(jointNumber,
jointOrientations);
           }// end if
           else
           { System.err.println("Wrong marker type (not 99)");
           }// end else

           }// end try
           catch(FieldNotFoundException fnfe)
             {
               System.err.println("Field not found! " + fnfe);
             }// end catch

           return;
         }

           /**
            * Invoked when an entity is modified.  The entity's articulation
information
            * is loaded into an ADUData packet, which is then serialized and
passed
            * to the Entity Master.
            */
           public void entityChanged(Entity e, Object info)
           {
           if(e instanceof DBPArticulatedGhost) return;

             ADUData articulationADU = null;
             articulationADU = new ADUData(new
       ADU(hiResDabpProtocol.getSchemaNamed("ARPPPDU")));

             if(resolution == 1)
             { articulationADU = new ADUData(new
       ADU(loResDabpProtocol.getSchemaNamed("ARPPPDU")));
             }

             // orientation
             double[] jointOpsArray = new double[4];

             Quat4d jointOps = new Quat4d();
             ((DBPArticulated)e).getJointOrientation(jointOps, jointNumber);
             jointOps.get(jointOpsArray);

             double jox = jointOpsArray[0];
             double joy = jointOpsArray[1];
             double joz = jointOpsArray[2];
             double jow = jointOpsArray[3];

             try
             {
             // load the ADUData packet
               articulationADU.put("packetResolution.resolution", new
       UnsignedByte(resolution));

               String tempString = new String("jointOrientations.x");
               if(articulationADU.get(tempString) instanceof DoublePrecision)
                 articulationADU.put(tempString, new DoublePrecision(jox));
```

```
              else if(articulationADU.get(tempString) instanceof
SinglePrecision)
                articulationADU.put(tempString, new
SinglePrecision((float)jox));

              tempString = new String("jointOrientations.y");
              if(articulationADU.get(tempString) instanceof DoublePrecision)
                articulationADU.put(tempString, new DoublePrecision(joy));
              else if(articulationADU.get(tempString) instanceof
SinglePrecision)
                articulationADU.put(tempString, new
SinglePrecision((float)joy));

              tempString = new String("jointOrientations.z");
              if(articulationADU.get(tempString) instanceof DoublePrecision)
                articulationADU.put(tempString, new DoublePrecision(joz));
              else if(articulationADU.get(tempString) instanceof
SinglePrecision)
                articulationADU.put(tempString, new
SinglePrecision((float)joz));

              tempString = new String("jointOrientations.w");
              if(articulationADU.get(tempString) instanceof DoublePrecision)
                articulationADU.put(tempString, new DoublePrecision(jow));
              else if(articulationADU.get(tempString) instanceof
SinglePrecision)
                articulationADU.put(tempString, new
SinglePrecision((float)jow));

            }// end try
            catch(FieldTypeException fte)
            {
              System.err.println("Field Type not found: " + fte);
            }// end catch
            catch(FieldNotFoundException fnfe)
            {
              System.err.println("Field not found: " + fnfe);
            }// end catch


      //      System.out.println("Send side ---
      ArticulationProtocol:entityChanged():articulationADU == \n" +
      articulationADU);//*************888888888888888888

            // Serialize and send articulation data for the entity
            byte[] data = serializeADU(articulationADU);

            Set distroSet = AOIM.getAOIM().getDistributionSet(e, this);

            EntityDispatcher.getEntityDispatcher().sendPacket(this,
                                                              e.getID(),
                                                              distroSet,
                                                              data);
          }

        /**
         * Sets the joint number that this protocol is for
         */
        public void setJointNumber(int pJoint)
        { jointNumber = pJoint;
        }// end setJointNumber()

        /**
```

```
   * Gets the joint number that this protocol is for
   */
  public int getJointNumber()
  { return jointNumber;
  }// end getJointNumber()
};
```

C. DBP ARTICULATION HIGH RESOLUTION SOURCE DOCUMENT

```
<?xml version="1.0"?>

<!DOCTYPE Protocol SYSTEM "./DynamicBehaviorProtocol.dtd">

<Protocol name="ARTICULATION_STATE"
          markerPosition="0"
          markerType="org.web3d.vrtp.datatypes.UnsignedByte"
          clientStub="http://some.url.com/baz">

<Type name="org.web3d.vrtp.datatypes.DoublePrecision"
value="file://org.web3d.vrtp.datatypes.DoublePrecision"></Type>

<Type name="org.web3d.vrtp.datatypes.UnsignedByte"
value="file://org.web3d.vrtp.datatypes.UnsignedByte"></Type>

<Type name="org.web3d.vrtp.datatypes.SignedInteger"
value="file://org.web3d.vrtp.datatypes.SignedInteger"></Type>

<!-- <Channel multicastAddress="225.7.8.30"
                multicastPort = "4070" /> -->

<Channel multicastAddress="225.8.7.1222"
                multicastPort = "2020" />

<PacketHeader name="RTP">

  <Field name="sequence"
         type="org.web3d.vrtp.datatypes.SignedInteger"
         initialValue="0"/>

  <Field name="timestamp"
         type="org.web3d.vrtp.datatypes.SignedInteger"
         initialValue="0"/>

</PacketHeader>

<ADU name="ARPPPDU" markerValue="95">

<!--  <Field name="markerValue"
          type="org.web3d.vrtp.datatypes.UnsignedByte"
          initialValue="95"/> -->

<Structure name="packetResolution">

          <Field name="resolution"
          type="org.web3d.vrtp.datatypes.UnsignedByte"
          initialValue="2"/>

</Structure>

<Structure name="jointOrientations">

      <Field name="x"
```

```
                type="org.web3d.vrtp.datatypes.DoublePrecision"
                initialValue="0.0"/>

        <Field name="y"
                type="org.web3d.vrtp.datatypes.DoublePrecision"
                initialValue="0.0"/>

        <Field name="z"
                type="org.web3d.vrtp.datatypes.DoublePrecision"
                initialValue="0.0"/>

        <Field name="w"
                type="org.web3d.vrtp.datatypes.DoublePrecision"
                initialValue="0.0"/>

</Structure>

</ADU>

</Protocol>
```

D. DBP ARTICULATION LOW RESOLUTION SOURCE DOCUMENT

```
<?xml version="1.0"?>

<!DOCTYPE Protocol SYSTEM "./DynamicBehaviorProtocol.dtd">

<Protocol name="ARTICULATION_STATE"
          markerPosition="0"
          markerType="org.web3d.vrtp.datatypes.UnsignedByte"
          clientStub="http://some.url.com/baz">

<Type name="org.web3d.vrtp.datatypes.SinglePrecision"
value="file://org.web3d.vrtp.datatypes.SinglePrecision"></Type>

<Type name="org.web3d.vrtp.datatypes.UnsignedByte"
value="file://org.web3d.vrtp.datatypes.UnsignedByte"></Type>

<Type name="org.web3d.vrtp.datatypes.SignedInteger"
value="file://org.web3d.vrtp.datatypes.SignedInteger"></Type>

<!-- <Channel multicastAddress="225.7.8.30"
                multicastPort = "4070" /> -->

<Channel multicastAddress="225.8.7.1222"
                multicastPort = "2020" />

<PacketHeader name="RTP">

  <Field name="sequence"
         type="org.web3d.vrtp.datatypes.SignedInteger"
         initialValue="0"/>


  <Field name="timestamp"
         type="org.web3d.vrtp.datatypes.SignedInteger"
         initialValue="0"/>

</PacketHeader>

<ADU name="ARPPPDU" markerValue="95">

<!--  <Field name="markerValue"
```

```
            type="org.web3d.vrtp.datatypes.UnsignedByte"
            initialValue="95"/> -->

      <Structure name="packetResolution">

            <Field name="resolution"
            type="org.web3d.vrtp.datatypes.UnsignedByte"
            initialValue="1"/>

      </Structure>

      <Structure name="jointOrientations">

            <Field name="x"
                  type="org.web3d.vrtp.datatypes.SinglePrecision"
                  initialValue="0.0"/>

            <Field name="y"
                  type="org.web3d.vrtp.datatypes.SinglePrecision"
                  initialValue="0.0"/>

            <Field name="z"
                  type="org.web3d.vrtp.datatypes.SinglePrecision"
                  initialValue="0.0"/>

            <Field name="w"
                  type="org.web3d.vrtp.datatypes.SinglePrecision"
                  initialValue="0.0"/>

      </Structure>

      </ADU>

      </Protocol>
```

E. SUMMARY

The source code and source documents presented in this appendix work together to provide both a high and low resolution capability for the DBP articulation protocol.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX F

A. INTRODUCTION

This appendix shows how to produce an XML source document for an example dynamic protocol created using an X3D Edit DBP Protocol editor.

B. DBP PROTOCOL EDITOR

An XML editor for DBP patterned after X3D Edit (Web 3D Consortium, 01), authored by Don Brutzman, can be used to generate DBP protocols for use in NPSNET-V. This DBP-Edit editor enables a user to create protocols using a symbolic GUI interface, rather than requiring the tedium of typing the protocols.

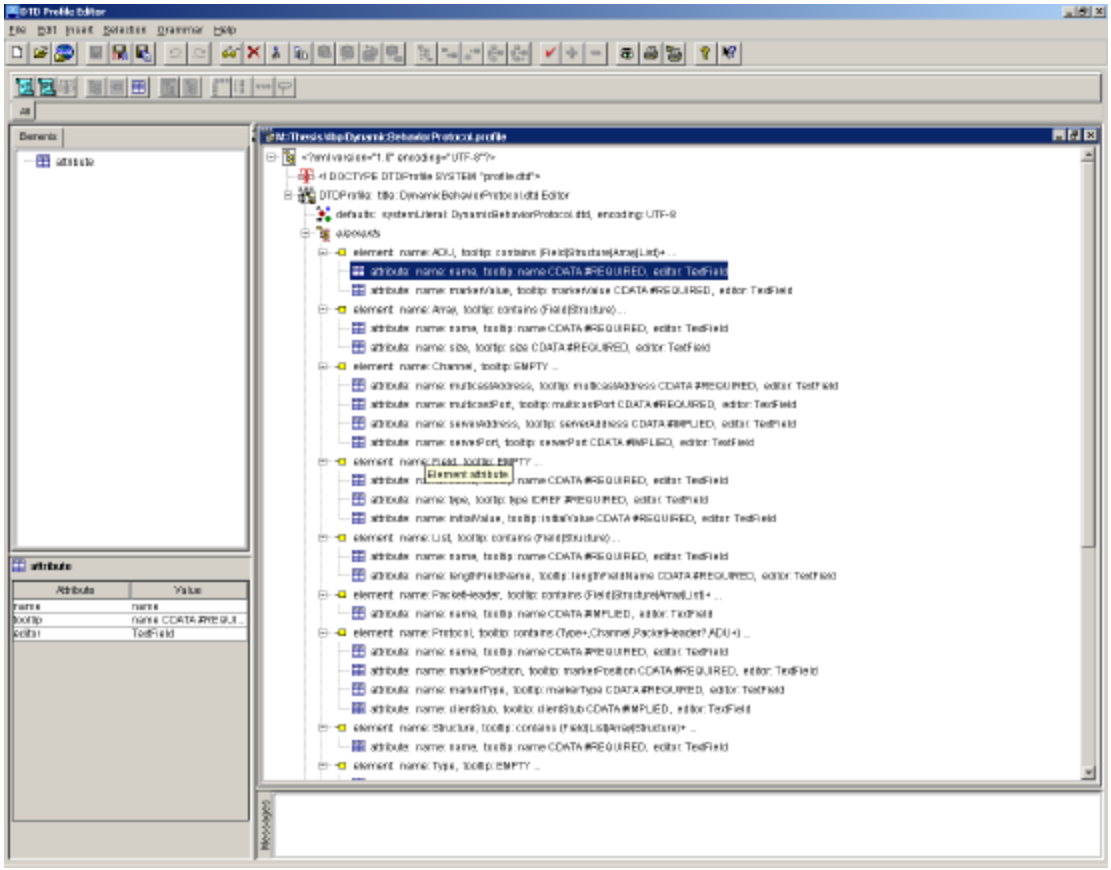

Figure F.1 DBP profile editor to modify tooltips for elements and attributes

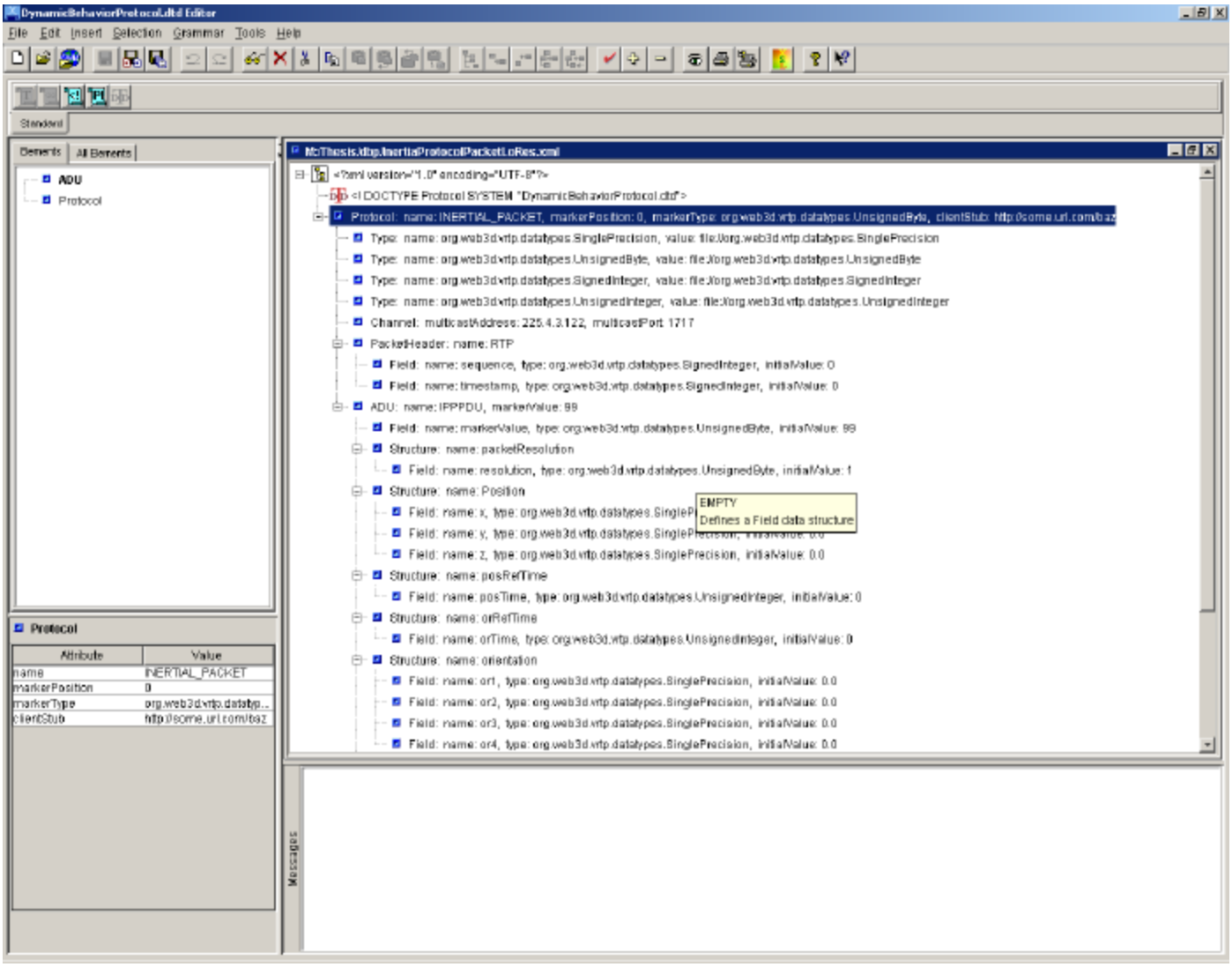

Figure F.2 DBP-Edit interface

C. DBP DATA-TYPE DEFINITION (DTD)

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- A dtd for a flexible protocol description language  -->

<!-- Author: Don McGregor -->
<!-- Date: 3/13/2000-->
<!---->
<!-- The protocol tag encloses some descriptions about the protocol-->
<!-- itself, followed by the ADUs in the protocol. The + sign means-->
<!-- one or more, the ? means zero or one.                          -->

<!ELEMENT Protocol ((Type+), Channel, (PacketHeader?), (ADU+))>
```

```
      <!-- Attributes: name: The name of the protocol (dis, etc.)         -->
      <!--             markerPosition: the number of bytes to offset to look
for the field -->
      <!--                            that distinguishes between different
ADUs in the    -->
      <!--                            protocol.                           -->
      <!--             markerType: The type of the data that distinguishes
between ADUs.   -->
      <!--                         This must be one of the listed Types below.
byte, short -->
      <!--                         int, etc.                              -->
      <!--             clientStub: The stub program we use to tie syntax to
semantics.     -->

      <!ATTLIST Protocol  name           CDATA #REQUIRED
                          markerPosition CDATA #REQUIRED
                          markerType     CDATA #REQUIRED
                          clientStub     CDATA #IMPLIED>

      <!-- A type is just a string that describes a URL. The thing at that
location    -->
      <!-- has code for defining the type of data. These are the basic building
blocks -->
      <!-- of the protocol; how we bootstrap up an initial set of basic types,
like    -->
      <!-- int, short, double, etc.                                        -->

      <!ELEMENT Type EMPTY>

      <!-- Attributes: The type name is an ID, which means it can be referenced
-->
      <!-- elsewhere in the XML file.                                      -->

      <!ATTLIST Type name ID #REQUIRED
                     value CDATA #REQUIRED>

      <!-- A MulticastGroup is just an element that can hold things like
multicast     -->
      <!-- port, etc                                                       -->

      <!ELEMENT Channel EMPTY >

      <!-- Attributes: address: mcast address used by this protocol.      -->
      <!--             port: mcast port to listen to for packets          -->

      <!ATTLIST Channel        multicastAddress CDATA #REQUIRED
                               multicastPort    CDATA #REQUIRED
                               serverAddress    CDATA #IMPLIED
                               serverPort       CDATA #IMPLIED>

      <!-- An ADU contains tags for various types of fields. The ADU tag must
contain -->
      <!-- at least one field tag - a field, structure, Array, or List.   -->


      <!ELEMENT ADU ((Field | Structure | Array | List)+)>
      <!ATTLIST ADU name CDATA #REQUIRED
                     markerValue CDATA #REQUIRED>

      <!-- Field types, which are slightly tricky. There are four basic types
of      -->
      <!-- fields: a primitive type, which describes something like a byte
-->
```

```
      <!-- or a float; a structured type, which can contain a several primitive
types;-->
      <!-- an array, which has a fixed length repetition of Field or Structure;
-->
      <!-- and a List, which has a variable length (per-packet) number or
repeitions  -->
      <!-- of a Field or a Structure.-->

      <!-- The basic Field type, which describes one of the Types listed above.
-->
      <!-- This is an empty tag - all the information is contained in the tag
and       -->
      <!-- its attributes, not in anything the start and end tags surround. In
-->
      <!-- fact, you can't have anything between the tags - it's required to be
-->
      <!-- be empty, usually done with <Field name="foo" type="byte"
initialValue="0"/> -->

      <!ELEMENT Field EMPTY>

      <!-- Attributes: name: Name for the field, such as "x" or "speed"
-->
      <!--             type: basic datatype, must be from Type list above
-->
      <!--             initialValue: Default value for field , such as "0" or
"17".  -->

      <!ATTLIST Field name CDATA #REQUIRED
                      type IDREF #REQUIRED
                      initialValue CDATA #REQUIRED>

      <!-- The Structure tag defines an enclosing container for other fields.
This  -->
      <!-- lets us build up larger constructs, such as "Point" that contains
three -->
      <!-- Fields, (x,y,z). or we could have a Strucutre that contains a
mixture   -->
      <!-- of Fields, lists, and other structures.                      -->

      <!ELEMENT Structure ((Field | List | Array | Structure)+)>

      <!-- Attributes: name: the name of the structure.                 -->

      <!ATTLIST Structure name CDATA #REQUIRED>

      <!-- Information about the PacketHeader. This is intended to describe,
for     -->
      <!-- example, the RTP header for a packet. This is prepended to every
outgoing -->
      <!-- payload, and used for routing, etc. This is really just an ADU
undercover -->

      <!-- The Array tag must contain either ONE Field tag or ONE Structure
tag.   -->
      <!-- This piece of data is repeated some number of times. Under Array,
-->
      <!-- this is a number that does not change, ever. If you say it has five
-->
      <!-- elements, it will always have space for five elements.
-->
```

```
      <!-- The Array, which is always some parse-time defined length.         -
->

      <!ELEMENT Array (Field | Structure)>

      <!-- Attributes:  name: Name of array                                  -->
      <!--              size: Number of times the element contained repeats.
-->
      <!--                    must always be the same; cannot change after
the   -->
      <!--                    document is parsed.                          -->

      <!ATTLIST Array      name CDATA #REQUIRED
                           size CDATA #REQUIRED>

      <!-- The List element has a variable length, which may change from packet
-->
      <!-- to packet. The length of the list is defined in some other field in
-->
      <!-- the payload. We save a reference to that field as an attribute; at
-->
      <!-- runtime, we lookup the value of the field to determine how long the
-->
      <!-- list will be in this instance. A List repeats either a primitive
Field  -->
      <!-- or a Structure zero or more times.                               -->

      <!ELEMENT List (Field | Structure)>

      <!-- Attributes: Name: Name of the list                               -->
      <!--             lengthFieldName: name of the field in this payload that
tells-->
      <!--             us how long this list is.                           -->

      <!ATTLIST List name CDATA #REQUIRED
                     lengthFieldName CDATA #REQUIRED>

      <!ELEMENT PacketHeader ((Field | Structure | Array | List)+)>

      <!-- Attributes: name: name of the packet header, eg 'rtp'.          -->

      <!ATTLIST PacketHeader name CDATA #IMPLIED>
```

D. DBP PROFILE

```
      <?xml version="1.0" encoding="UTF-8"?>
      <!DOCTYPE DTDProfile SYSTEM "profile.dtd">
      <DTDProfile title="DynamicBehaviorProtocol.dtd Editor">
        <defaults encoding="UTF-8"
systemLiteral="DynamicBehaviorProtocol.dtd"/>
        <elements>
          <element name="ADU" tooltip="contains
(Field|Structure|Array|List)+&#10;Application Data Unit (ADU) definition.">
            <attribute editor="TextField" name="name" tooltip="name CDATA
#REQUIRED"/>
            <attribute editor="TextField" name="markerValue"
tooltip="markerValue CDATA #REQUIRED"/>
          </element>
          <element name="Array" tooltip="contains (Field|Structure)&#10;defines
an Array of Fields or Structures">
```

```
            <attribute editor="TextField" name="name" tooltip="name CDATA
#REQUIRED"/>
            <attribute editor="TextField" name="size" tooltip="size CDATA
#REQUIRED"/>
          </element>
          <element name="Channel" tooltip="EMPTY&#10;Defines multicast channel
parameters">
            <attribute editor="TextField" name="multicastAddress"
tooltip="multicastAddress CDATA #REQUIRED"/>
            <attribute editor="TextField" name="multicastPort"
tooltip="multicastPort CDATA #REQUIRED"/>
            <attribute editor="TextField" name="serverAddress"
tooltip="serverAddress CDATA #IMPLIED"/>
            <attribute editor="TextField" name="serverPort" tooltip="serverPort
CDATA #IMPLIED"/>
          </element>
          <element name="Field" tooltip="EMPTY&#10;Defines a Field data
structure">
            <attribute editor="TextField" name="name" tooltip="name CDATA
#REQUIRED"/>
            <attribute editor="TextField" name="type" tooltip="type IDREF
#REQUIRED"/>
            <attribute editor="TextField" name="initialValue"
tooltip="initialValue CDATA #REQUIRED"/>
          </element>
          <element name="List" tooltip="contains (Field|Structure)&#10;Defines
a list of Fields or Structures">
            <attribute editor="TextField" name="name" tooltip="name CDATA
#REQUIRED"/>
            <attribute editor="TextField" name="lengthFieldName"
tooltip="lengthFieldName CDATA #REQUIRED"/>
          </element>
          <element name="PacketHeader" tooltip="contains
(Field|Structure|Array|List)+&#10;defines PacketHeader contents">
            <attribute editor="TextField" name="name" tooltip="name CDATA
#IMPLIED"/>
          </element>
          <element name="Protocol" tooltip="contains
(Type+,Channel,PacketHeader?,ADU+)&#10;Root element to define a Dynamic
Behavior Protocol.">
            <attribute editor="TextField" name="name" tooltip="name CDATA
#REQUIRED"/>
            <attribute editor="TextField" name="markerPosition"
tooltip="markerPosition CDATA #REQUIRED"/>
            <attribute editor="TextField" name="markerType" tooltip="markerType
CDATA #REQUIRED"/>
            <attribute editor="TextField" name="clientStub" tooltip="clientStub
CDATA #IMPLIED"/>
          </element>
          <element name="Structure" tooltip="contains
(Field|List|Array|Structure)+&#10;defines a data Structure">
            <attribute editor="TextField" name="name" tooltip="name CDATA
#REQUIRED"/>
          </element>
          <element name="Type" tooltip="EMPTY&#10;defines a payload data
structure type">
            <attribute editor="TextField" name="name" tooltip="name ID
#REQUIRED"/>
            <attribute editor="TextField" name="value" tooltip="value CDATA
#REQUIRED"/>
          </element>
          <element name="XML_COMMENT" tooltip="Comment -&#10;Use it anywhere to
make the document clearer."/>
```

```
          <element name="XML_TEXT" tooltip="TEXT"/>
          <element name="XML_CDATA" tooltip="CDATA Section -&#10;Use it to
store marked-up text that you want to store as data."/>
          <element name="XML_PI" tooltip="Processing Instruction (PI) -&#10;Use
it to pass directions and information to programs."/>
          <element name="XML_DOCUMENT_TYPE" tooltip="Document Type."/>
        </elements>
        <documents/>
        <palettes>
          <toolbarpalette title="Standard">
            <group elements="XML_TEXT XML_CDATA XML_COMMENT XML_PI
XML_DOCUMENT_TYPE"/>
          </toolbarpalette>
          <sidebarpalette all-visible="no" searchable="no" title="Elements">
            <group elements="ADU Array Channel Field List PacketHeader Protocol
Structure Type"/>
          </sidebarpalette>
          <sidebarpalette all-visible="yes" searchable="no" title="All
Elements">
            <group elements="ADU Array Channel Field List PacketHeader Protocol
Structure Type"/>
          </sidebarpalette>
        </palettes>
        <importers/>
        <exporters/>
        <tools>
          <tool class-name="com.ibm.hrl.xmleditor.extension.xsl.XSLHandler"/>
        </tools>
      </DTDProfile>
```

E. SUMMARY

The documents presented in this appendix provide a DBP protocol editor that uses X3D Edit to generate protocols in XML, and an example dynamic protocol for use in NPSNET-V.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Afonso, Francisco, *Virtual Reality Transfer Protocol (VRTP): Implementing a Monitor Application for the Real-Time Transport Protocol (RTP) Using the Java Media Framework (JMF)*, Masters Thesis, Naval Postgraduate School, Monterey California September 1999.

Alexander, Christopher and others, A Pattern Language, Oxford University Press, New York, 1977.

Apache http server, The Apache Software Foundation, 2001. Information available at: http://www.apache.org/.

Brutzman, Don, Graphics Internetworking: Bottlenecks and Breakthroughs, Addison-Wesley, 1977.

Brutzman, Don, Zyda, Mike, Watsen, Kent and Macedonia, Mike,"virtual reality transfer protocol (vrtp) Design Rationale," *Workshops on Enabling Technology: Infrastructure for Collaborative Enterprises (WET ICE): Sharing a Distributed Virtual Reality*, Massachusetts Institute of Technology, Cambridge Massachusetts, June 18-20 1997.

Capps, Michael and McGregor, Don, NPSNET-V, 2000. Information available at: http://npsnet.org/NPSNET-V/npsnetV-cga.pdf .

Eudora, LDAP Freeware Products, 2001. Information available at: http://www.eudora.com/free/ldap.html.

Gamma, Erich and others, Design Patterns, Addison Wesley Longman, Inc., 1995.

Grand, Mark, Patterns in Java Volume 1, John Wiley & Sons, Inc., 1998.

Harold, Elliotte Rusty, XML Bible, IDG Worldwide Books Inc. 1999.

Jain, Raj, The Art of Computer Systems Performance Analysis, John Wiley & Sons, Inc. 1991.

McGregor, D. NPSNET-V, 2001. Information available at: http://www.npsnet.org/~npsnet/v/index.html.

McKusick, Marshall Kirk and others, The Design and Implementation of the 4.4 BSD [Unix] Operating System, Addison Wesley Publishing Company, Inc., 1996.

Peterson, L. and Davie, B., Computer Networks, Morgan Kaufman Publishers, 2000.

Singhal, S. and Zyda, M., Networked Virtual Environments, Addison-Wesley Publishing Company, Inc., 1999.

Stevens, W. Richard, TCP/IP Illustrated, Volume 1, Addison Wesley Publishing Company, Inc., 1994.

Sun, The Source for Java[TM] Technology, 2001. Information available at: http://java.sun.com/.

Wathen, Michael, Dynamic Scalable Network Area of Interest Management for Virtual Worlds, Naval Postgraduate School, 2001.

Web 3D Consortium, Extensible 3D (X3D) Graphics Working Group, 2001. Information available at: http://www.web3d.org/x3d.html.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center.....2
   8725 John J. Kingman Road, Suite 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library.....2
   Naval Postgraduate School
   411 Dyer Road
   Monterey, CA 93943-5101

3. Dr. Michael P. Bailey.....1
   Technical Director, Marine Corps Training and Education Command
   Commanding General
   Marine Corps Combat Development Command, Code 46T
   3300 Russell Road
   Quantico, VA 22134

4. Dr. Philip S. Barry.....1
   Chief, S&T Initiatives Division
   Defense Modeling and Simulation Office
   1901 N. Beauregard Street, Suite 500
   Alexandria VA 22311

5. Robert J Barton III.....1
   Fraunhofer Center for Research in Computer Graphics (CRCG)
   321 South Main St
   Providence, RI 02903

6. Curtis Blais, Code IJWA.....1
   Naval Postgraduate School
   Monterey, CA 93940-5000

7. Gordon, Bradley, Code OR.....1
   Naval Postgraduate School
   Monterey, CA 93940-5000

8. Don Brutzman, Code UW/Br.....10
   Naval Postgraduate School
   Monterey, CA 93940-5000

9. Dan Boger, Code C4I 1
Naval Postgraduate School
Monterey, CA 93940-5000

10. Alex Bordetsky, Code IS 1
Naval Postgraduate School
Monterey, CA 93943-5101

11. Rex Buddenberg Code IS/Bu 1
Naval Postgraduate School
Monterey, CA 93940-5000

12. Fred Burkley 1
NAVSEA Undersea Warfare Center
Division Newport
Code 2231, Bldg 1171-3
1176 Howell Street
Newport, RI 02841-1708

13. Bob Cabanya 1
Information Operations, Inc.
1298 Bay Dale Dr.
Arnold, MD 21012

14. LtCol Neil Cadwallader, USMC 1
MCTSSA, Box 555171
Camp Pendelton, CA 92055-5171

15. Erik Chaum 1
NAVSEA Undersea Warfare Center
Division Newport
Code 2231, Building 1171-3
1176 Howell Street
Newport, RI 02841-1708

16. Raymond Christian 1
Submarine Sonar Code 21
1141 Howell Street
Newport, RI 02841

17. Robert Clover 1
Institute for Defense Analyses
1801 N. Beauregard St.
Alexandria, VA 22311-1772

18. David Colleen 1
Planet 9 Studios
753 Kansas Street
San Francisco, CA 94107

19. George Conner, Code 01 1
Naval Postgraduate School
Monterey, CA 93943-5101

20. Justin Couch, Alan Hudson, and Stephen Matsuba 1
Yumatech, Inc
600 Malden Ave. East
Suite 202
Seattle, WA 98112

21. Colonel William Crain, USA 1
Defense Modeling and Simulation Office
1901 N. Beauregard St. Suite 500
Alexandria, VA 22311

22. Paul Diefenbach 1
Open Worlds OpenWorlds, Inc.
3508 Market Street
Suite 203
Philadelphia, PA. 19104

23. Maj T.J. Ferrell, USMC 1
MCMSMO
Marine Corps Combat Development Command, Code 46T
3300 Russell Road
Quantico, VA 22134

24. Dr. Paul Fishwick 1
Computer & Information Science and Engineering Department
University of Florida
Post Office Box 115120
322 Building CSE
Gainsville, FL 32611-6120

25. CAPT Donald Gerry, USN 1
Chief Staff Officer
COMSUBDEVRON TWELVE
P.O. Box 70, NSB New London
Groton, CT 06340

26. Simon Goerger, Code 32 1
Naval Postgraduate School
Monterey, CA 93943-5101

27. Dr. Tony Healey, Code ME 1
Naval Postgraduate School
Monterey, CA 93943-5101

28. John Hiles 1
Naval Postgraduate School
Monterey, CA 93943-5101

29. Doug Horner, Code C4I 1
Naval Postgraduate School
Monterey, CA 93943-5101

30. Wayne Hughes, Code OR 1
Naval Postgraduate School
Monterey, CA 93943-5101

31. Captain Mike Hunsberger, USAF 1
Air Force Communications Agency/TCPD
Scott AFB, IL, 62225

32. Jerry Isdale 1
HRL Laboratories
3011 Malibu Canyon Road
Malibu, CA 90265-4797

33. Pamela Krause 1
Advance Systems & Technology
National Reconnaissance Office
14675 Lee Rd
Chantilly, VA 20151-1714

34. S. David Kwak 1
The Mitre Corporation – M/S B155
202 Burlington Rd.
Bedford, MA 01730-1420

35. John Lademan 1
Electronic Sensors and Systems Sector
Northrop Grumman Corporation
PO Box 1488 – MS 9030
Anapolis, MD 21404

36. Major Dave Laflam, USA 1
Army Model and Simulation Office
Office of the Deputy Chief of Staff for Operations and Plans
1111 Jefferson Davis Highway
Crystal Gateway North (Suite 503E)
Arlington, VA 22202

37. Dr. Francisco Loaiza and Dr. Eugene Simaitis 1
Institute for Defense Analyses
Systems Evaluation Division
1801 N. Beauregard St.
Alexandria, VA 22311

38. Dr. R Bown Loftin 1
Director of Simulation Programs
Virginia Modeling Analysis & Simulation Center
Old Dominion University
7000 College Dr
Suffolk, VA 23435

39. CAPT Arnold O. Lotring, USN 1
Strategic Study Group
Naval War College
686 Cushing Rd.
Newport, RI 02841

40. Dell Lunceford 1
Director, Army Model & Simulation Office
Crystal Gateway North Suite 503E
1111 Jefferson Davis Highway
Arlington, VA 22202

41. Mike Macedonia 1
Chief Scientist and Technical Director
US Army STRICOM
12350 Research Parkway
Orlando, FL 32826-3276

42. Fahrid Mamaghani 1
19223 SE 45th St
Issaquah, WA 98027

43. CAPT Robert Maslowsky, USN 1
CNO/N62
Pentagon
Washington, D.C. 20350-2000

44. Michael McCann.....1
Monterey Bay Aquarium Research Institute (MBARI)
PO Box 628
Moss Landing, CA 95039-0628

45. Chuck Mirabile 1
USMC Program Office, D12
52560 Hull St.
San Diego, CA
92152-5001

46. Dr. Katherine L. Morse 1
Chief Scientist
Director, Modeling & Simulation Business Unit
Epsilon Systems Solutions
2550 Fifth Ave., Ste. 725, San Diego, CA 92103

47. Capt Mark Murray USAF 1
Joint Battlespace Infosphere (JBI)
AFRL/IFSE
Building 3, Room E-1078
525 Brooks Road
Rome, NY 13441-4505

48. Michael Myjak 1
Vice President and CTO
The Virtual Workshop
PO Box 98
Titusville, FL 32781

49. Rick Nelson.....1
OpenWorlds, Inc.
3508 Market Street
Suite 203
Philadelphia, PA. 19104

50. Captain Shane Nicklaus, USMC 1
503 Ross Circle
Martinez, CA 94553

51. Neal Park, President 1
Nexternet, Inc.
2900 Gordon Ave.
Suite 202
Santa Clara, CA 95051

52. Marty Paulsen 1
Analytic Graphics, Inc.
3760 Killroy Airport Way
Suite 270
Long Beach, CA 90806

53. CPT Joel Palowski, USA 1
TRAC-Monterey
Naval Postgraduate School
PO Box 8692
Monterey, CA 93943

54. George Philips 1
CNO, N6M1
2000 Navy Pentagon
Room 4C445
Washington, DC 20350-2000

55. Dr. David Pratt 1
SAIC
12479 Research Parkway
Orlando, FL 32826-3248

56. Dr. Mark Pullen & Dr. Robert Simon 2
Department of Computer Science/C3I Center MS4A5
George Mason University
FairFax, VA 22030

57. Dick Puk 1
President
Intelligraphicas Incorporated
7644 Cortina Court
Carlsbad, CA 92009-8206

58. CAPT Jason Quigley,USAF 1
Joint Battlespace Infosphere (JBI)
AFRL/IFSE
Building 3, Room E-1078
525 Brooks Road
Rome, NY 13441-4505

59. Dr. Martin Reddy 1
SRI International, EK219
333 Ravenswood Avenue
Menlo Park, CA 94025

60. Nancy Roberts, Code SM 1
Naval Postgraduate School
Monterey, CA 93943-5101

61. Dr. R. Jay Roland, President 1
Rolands and Associates
500 Sloat Avenue
Monterey CA 93940

62. Clyde Scandrett, Code MA 1
Naval Postgraduate School
Monterey, CA 93943-5101

63. Dr. John Sirmalis 1
NUWC
1176 Howell Street
Newport, RI 02841-1708

64. CDR Amy Smith, USN 1
614 Sicard Street, S.E.
Washington Navy Yard
(Bldg 201, 4E458)
Washington, DC 20376

65. John Stewart 1
3701 Carling Ave, Box 11490, Station H
Ottawa, Ontario K2H 8S2

66. RADM Paul Sullivan, USN 1
Director, Submarine Warfare Division N77
2000 Navy Pentagon, 4D542
Washington, DC 20350-2000

67. Craig Swanson 1
Science Applications International Corporation
Information Systems Division
1710 SAIC Dr.
Mclean, VA 22102

68. Dr. Ralph Toms 1
SRI International
333 Ravenswood Ave
Menlo Park, CA 94025

69. CAPT Robert Voigt, USN 1
Chair, Electrical Engineering Department
U.S. Naval Academy
Annapolis, MD 21402

70. Jeffrey Weekley 1
Rolands and Associates
500 Sloat Avenue
Monterey CA 93940

71. Joe Williams 1
3421 Bonita Vista Lane
Santa Rosa, CA 95404

72. LtCol Paul Ziegenfuss 1
HQMC C4I Plans and Policy Division
2 Navy Annex
Washington, D.C. 20380-1775

73. Walter H. Zimmers 1
Defense Threat Reduction Agency
CPOC
6801 Telegraph Road
Alexandria VA 22310-3398

74. Dr. Michael Zyda, Code MOVES 1
Chair, Modeling Virtual Environments and Simulation Academic Group
Naval Postgraduate School
Monterey, CA 93940-5000